

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación



Estudio de las redes definidas por software y escenarios virtuales de red orientados al aprendizaje

*Study of software defined networks and learning-oriented virtual network
environments.*

TRABAJO FIN DE MÁSTER

Javier Cano Moreno

2015

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

**Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos**

TRABAJO FIN DE MÁSTER

**Estudio de las redes definidas por software y
escenarios virtuales de red orientados al
aprendizaje**

*Study of software defined networks and learning-oriented virtual network
environments.*

Autor

Javier Cano Moreno

Director

David Fernández Cambronero

2015

RESUMEN

La evolución de la gestión de las redes de comunicaciones está actualmente sufriendo un gran cambio debido al uso de nuevas arquitecturas y formas de gestión que prometen ser más eficaces en muchos ámbitos. Además, la virtualización de cualquier servicio (sistemas, almacenamiento, etc.) es una técnica que desde sus principios ha mejorado mucho y está ya implementada en gran parte de los centros de datos de todo el mundo. No pasa igual con las redes de comunicaciones, donde desde hace muchos años existen algunas técnicas de virtualización pero hasta hace pocos años no se ha comenzado a hablar de la virtualización de las redes a un nivel parecido al que se habla con, por ejemplo, los sistemas.

Con estas ideas comienzan a surgir propuestas de cambio en la gestión de las redes de comunicaciones, comenzando con la separación del plano de datos y de control para conseguir una gestión centralizada con posibilidad de tener una visión global y real de toda la red. De estas propuestas y estándares el que por sus características ha resultado ser por el que se decantan los desarrolladores de soluciones de red es Openflow.

El estándar de Openflow implementa toda la gestión de una red centralizando ésta en los llamados controladores SDN, de forma que deja a los dispositivos de red (switchs) compatibles con Openflow únicamente la función que implementa tradicionalmente el plano de datos. De esta forma, los switchs encaminarán los diferentes flujos de tráfico en función de la información que tenga en su llamada tabla de flujos, la cual es gestionada por el controlador SDN. Estos controladores son típicamente soluciones software que implementan, en este caso, el estándar de Openflow (aunque no hay que pensar que sea el único a utilizar en estas arquitecturas) y que a través las diferentes interfaces que implementa, tanto los switchs podrán comunicarse de forma segura con él (*Southbound API*), como las aplicaciones (*Northbound API*).

El tener la posibilidad de que las aplicaciones de una forma fácil puedan acceder a la configuración de la red es un hecho rompedor que cambia por completo el concepto de administración de red que se tiene hoy en día. El dinamismo que nace con estas posibilidades de gestión es muy grande, así como la escalabilidad de las redes, un punto muy importante que has ahora era bastante difícil de abordar sobretodo en grandes redes.

Además, con este *boom* de las SDN no paran de salir herramientas software que ofrecen nuevas funcionalidades que ayudan a mejorar el uso de las redes así como su gestión. Es el caso de los proyectos como Flowvisor, que permite realizar particiones de la red no sólo por dispositivos sino también por reglas de flujo, u OpenVirteX, que es capaz de virtualizar la red en diferentes redes virtuales ofreciendo capacidades de gestión incluso de recuperación frente a fallos (caída de un enlace, por ejemplo).

Otro tipo de soluciones que también están apareciendo y que pretenden ir un paso más allá, son por ejemplo el proyecto ONOS, que se define como un sistema operativo de red (NOS) que ofrece más posibilidades que un controlador SDN u OpenDayLight, un controlador SDN que no solo ofrece soporte al estándar de Openflow, sino que está abierto a dar soporte a cualquier estándar de gestión de red de cualquier fabricante.

En este trabajo se va a presentar el estado actual de las arquitecturas SDN y se van a implementar algunos escenarios orientados al aprendizaje de las mismas. Así, con las herramientas de creación de escenarios de red virtuales como Mininet y VNX se crearán hasta tres escenarios con características diferentes que permitirán poner en práctica los conceptos

más importantes de las arquitecturas SDN, con el uso del controlador SDN Floodlight y las características que ofrece Flowvisor.

ABSTRACT

Networks administration world is nowadays facing big changes due to the implementation of new architectures and administration fashion with the goal of achieve more performance at different levels. Moreover, the virtualization in other domains, such as systems or storage, is a solution that have been improved a lot from it first implementation, in fact, the system&storage virtualization it's already present in most medium or big companies over the world. But it is not the case of networks, although there is longtime that various technologies exists and provide certain level of network virtualization, the evolution in this term is one or more order of magnitude below other areas.

In front of this *frozen* scenario several proposals begin to appear with the aim of improve network administration. Some of them goes for take away the control plane in order to centralize administration and keep the data plane on networks devices. This fashion allow you to have a full view of the network, so that, network administration procedures could be improved at less operating costs. Within these proposal there is one which have been accepted more than others, is the case of Openflow standard.

The Openflow standard implements the network administration centralizing it in components called *SDN controllers*. The control plane is migrated to these controllers while the data plane is, as for now, in network devices (switches). In this way, the switches will send flows to their controller and will wait until the controller say them what to do with this kind of flows. Next time the switch receive other similar flow, it will be treated like the previous one because the switch already have this entry in their flow table/s.

The controllers are typically software components that implements two main API's which permits connect to them. One of them is the *Southbound API*, which is consumed by networks devices to communicate with the controller. The second one is the *Northbound API*, which is offered from controllers to business applications in order to be able to manage the network from these applications.

This characteristic will allow to implement a dynamic network administration, which will allow more scalability and reduce a lot the time spent in network configuration and troubleshooting tasks. These to improvements, scalability and time, are the most important advantages that SDN and Openflow bring to us.

Moreover, there is a lot of software developments that help improve SDNs implementing new functionalities. It is the case of the projects like Flowvisor, which allow network partitioning based not only in network devices but in flows headers from level 1 to level 4. Other projects like OpenVirteX are interesting too being that is able to virtualize a physical network into several virtual networks independent of each other.

In this thesis I will explain the SDN architecture basis, then, several network scenarios will be implemented in order to help students to understand it. To implement these scenarios I will use tools like Mininet and VNX to create the virtual switches and hosts and with the help of Flowvisor and the SDN controller called Floodlight, we will learn and implement the main characteristics that these projects bring to us. These scenarios will be wrapped in different use cases which helps understand the use of this kind of architectures.

ÍNDICE

RESUMEN	1
ABSTRACT	3
ÍNDICE DE FIGURAS	5
INTRODUCCIÓN	6
Organización del documento	6
CAPÍTULO 1: OBJETIVOS Y PLAN DE DESARROLLO	7
Objetivos	7
Plan de desarrollo seguido	7
Medios	8
CAPÍTULO 2: ESTADO DEL ARTE	9
Arquitecturas SDN	11
Openflow	11
Interfaces	12
Controladores	13
Virtualización	21
Herramientas de creación de redes virtuales	30
Mininet	30
VNX: Virtual Networks over Linux	33
CAPÍTULO 3: ESCENARIOS	37
Herramientas utilizadas	37
Escenarios propuestos	37
Escenario 1	38
Escenario 2	40
Escenario 3	43
Configuración y pruebas	45
Escenario 1	46
Escenario 2	47
Escenario 3	49
CAPÍTULO 4: CONCLUSIONES Y TRABAJOS FUTUROS	51
Conclusiones	51
Futuros trabajos	51
BIBLIOGRAFÍA	53
ANEXO I	55
ANEXO II	61
ANEXO III	73

ÍNDICE DE FIGURAS

Fig. 1: Porcentaje interés de búsquedas web del término: SDN (en Google)	11
Fig. 2: Porcentaje interés de búsquedas web del término: Openflow (en Google)	11
Fig. 3: Arquitectura SDN – Interfaces y componentes	13
Fig. 4: Controladores open source - características	14
Fig. 9: Beacon overview [34]	15
Fig. 10: OpenDayLight overview.....	16
Fig. 5: Arquitectura Floodlight.....	19
Fig. 6: Floodlight web UI - Dashboard	19
Fig. 7: Floodlight web UI - Detalle de un switch.....	20
Fig. 8: Floodlight web UI - Topología de red	20
Fig. 11: FlowSpaceImpl.java - Bug Double NW_DST	25
Fig. 12: Particiones Flowvisor	28
Fig. 13: OVX – Arquitectura conceptual	29
Fig. 14: OVX - Detalle componentes.....	30
Fig. 15: miniedit - GUI experimental para Mininet	32
Fig. 16: VNX - Mapa escenario de red (ejemplo con VLANs).....	34
Fig. 17: Escenario 1.....	39
Fig. 18: Escenario 27.....	41
Fig. 19: Escenario 3.....	44
Fig. 20: Mapa red escenario 1 con VNX.....	46
Fig. 21: Mapa red Escenario 2 con VNX.....	48
Fig. 22: Tabla de flujos de switch S1 - Escenario 2	49
Fig. 23: Mapa red Escenario 3 con VNX.....	49
Fig. 24: Reglas de flujo para nuevo tráfico	49
Fig. 25: Conectividad mientras se cambia el camino que sigue el tráfico.....	50

INTRODUCCIÓN

Las redes de comunicación son hoy en día uno de los elementos básicos en cualquier entorno (empresarial o personal), y que su funcionamiento sea efectivo es una tarea prioritaria. Para que esto suceda así, debe haber una gestión de estas redes que sea también efectiva, y acorde con la gran evolución de todos los sistemas informáticos y los emergentes servicios en cloud que hoy en día se ofrecen. En estas líneas, el entorno de las redes está encontrando muchas dificultades a la hora de atender de forma flexible y rápida los continuos cambios y requisitos de los nuevos servicios que van apareciendo.

En primer lugar, el coste operacional de mantener grandes redes en funcionamiento, pasa por gestionar una gran cantidad de elementos de red (switchs, routers, etc.) de forma individual. Esta operativa penaliza los tiempos en los que una red puede adaptarse a los cambios (nuevos elementos, servicios con ciertos requisitos, reacción ante nuevas vulnerabilidades, etc.). Además, el conocimiento para llevar a cabo esta gestión debe ser muy alto y específico, lo cual también conlleva unos elevados costes operacionales.

Por otro lado, el nivel de innovación en las redes de comunicaciones es mucho menor al del resto de elementos que lo rodean. Las diferentes implementaciones de cada fabricante no hacen sino poner barreras a la evolución de las redes y de su adaptación a los servicios que más se demandan hoy en día.

Se ponen de manifiesto entonces una serie puntos débiles que aún a día de hoy no se han abordado de una forma efectiva. En temas de seguridad, capacidad de gestión, adaptación a los nuevos cambios, los sistemas de redes tradicionales necesitan un empuje que las haga más flexibles, escalables, fáciles de gestionar y securizar.

Debido a las carencias indicadas anteriormente se considera que la evolución de las redes definidas por software es el camino que ofrece las mejores soluciones, y por ello en este trabajo se estudia el caso de la arquitectura de una SDN con particionado de los flujos de tráfico, para lo cual se utilizará el software Flowvisor [10], que nos permitirá crear diferentes espacios de red en base a diferentes parámetros de las cabeceras de nivel 2, nivel 3 y nivel 4.

Organización del documento

A continuación se comentará el estado del arte de las arquitecturas SDN y de sus componentes. Tras esta revisión, se pasará al siguiente punto donde se detallarán los componentes utilizados en este trabajo (contenedores LXC, controlador SDN, Flowvisor y herramientas de simulación de escenarios de red).

Una vez conocidos todos los componentes elegidos, se expondrán algunos escenarios basados en arquitecturas SDN que servirán para tener una primera toma de contacto con estas tecnologías. Tras ello, se detallará otro escenario algo más complicado donde se explotan las ventajas que ofrece la herramienta de Flowvisor en conjunto con el resto de componentes de la arquitectura.

Finalmente se propondrán algunos supuestos y la arquitectura más adecuada a implementar de acuerdo a todo lo investigado en este proyecto, de forma que pueda servir de guía rápida a la hora de plantearse si es conveniente el uso de SDNs, Flowvisor u otras implementaciones.

CAPÍTULO 1: OBJETIVOS Y PLAN DE DESARROLLO

Objetivos

El primer objetivo es explorar el estado actual de las SDN y entender su funcionamiento. Para ello se consultará la información disponible, así se podrá conocer su historia, entender bien las necesidades que cubre y las razones por las cuales es interesante estudiar estas soluciones. Para esto será necesario ver comparativas entre distintas implementaciones, valorar los aspectos positivos y negativos de cada una de ellas, revisar las soluciones comerciales existentes hoy en día... y con ello se podrán plasmar los problemas existentes y las posibles soluciones que se pueden dar con este tipo de arquitecturas de red.

Como segundo objetivo se plantea el conocer a fondo los principales componentes y soluciones que implementan arquitecturas SDNs [2],[3] (controladores, emuladores de entornos de red, herramientas de despliegue, posible integración entre algunos o todos ellos, etc.)







El tercer objetivo importante es el de proponer algún escenario donde se puedan valorar y verificar lo aprendido en los puntos anteriores. Para ello se hará una elección de la/las arquitectura/s para implementar los escenarios basadas en diferentes criterios que se estimen importantes a la hora de implementar una solución basada en SDN. Un escenario interesante sería el que utiliza diferentes controladores a través de un *proxy* (como Flowvisor, por ejemplo) para la gestión de los flujos de red.

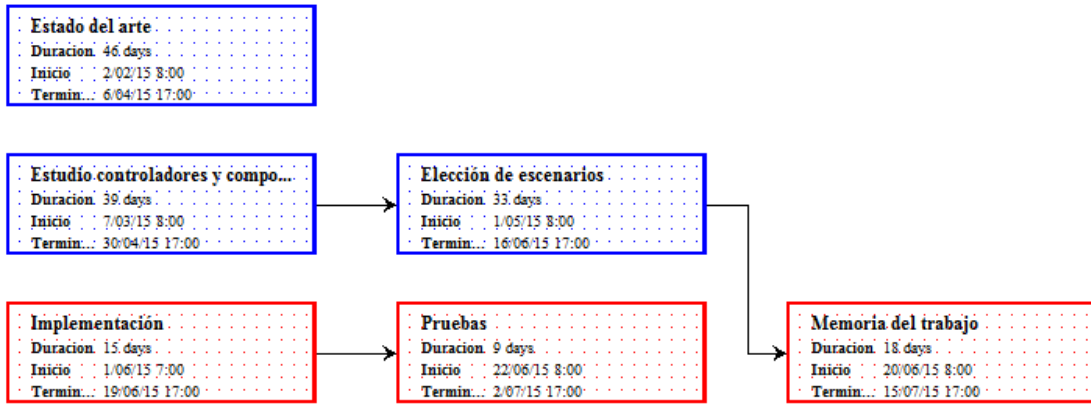
Plan de desarrollo seguido

El plan de desarrollo seguido está directamente ligado con la consecución de los objetivos mencionados anteriormente. De esta forma, el plan se ha desarrollado en 5 fases principales:

1. Estudio de las SDNs: Estado del arte y comprensión de conceptos.
2. Estudio detallados de controladores SDN, componentes para su implementación, herramientas de despliegue, etc.
3. Elección de escenarios, implementación y pruebas.
4. Propuesta de arquitecturas y componentes basada en diferentes necesidades y requisitos
5. Redacción de la memoria del trabajo fin de máster.

A continuación se presenta el plan de desarrollo indicando los tiempos dedicados en cada una de las fases:

	Nombre	Duracion	Inicio	Terminado
	Estado del arte	46 days	2/02/15 8:00	6/04/15 17:00
	Estudio controladores y componentes	39 days	7/03/15 8:00	30/04/15 17:00
	Elección de escenarios	33 days	1/05/15 8:00	16/06/15 17:00
	Implementación	15 days	1/06/15 7:00	19/06/15 17:00
	Pruebas	9 days	22/06/15 8:00	2/07/15 17:00
	Memoria del trabajo	18 days	20/06/15 8:00	15/07/15 17:00



Medios

Se ha utilizado un equipo de gama media (100Gb de disco, 4Gb de RAM, 2 CPU) para realizar todas las implementaciones y pruebas. Se ha utilizado como host un sistema operativo Ubuntu 14.04, sobre el que se implementan todas las pruebas y los escenarios propuestos. Se ha elegido esta plataforma como base puesto que es compatible con todas las implementaciones vistas hasta ahora, incluso algunas directamente utilizan o recomiendan este sistema operativo como base.

CAPÍTULO 2: ESTADO DEL ARTE

La virtualización de las redes está presente desde hace ya muchos años y es altamente utilizada casi todas las redes Ethernet. El uso de circuitos virtuales o de aislamiento de dominios de difusión (VLAN[1]), es una forma muy típica de uso de virtualización en los entornos de red. Otras técnicas que han ido apareciendo con el paso de los años, como MPLS [5], VPN[6]o VRFs [7], son también muy utilizadas, por ejemplo, en los *backbone* de los proveedores de servicios (ISPs) es muy normal encontrar redes que ofrecen servicios de *tunneling* de tráfico basado en MPLS, pudiendo establecer múltiples enlaces privados y aislados entre diferentes redes utilizando la misma infraestructura física.

No obstante, de cara a abordar esas debilidades descritas en la introducción, se comienzan a realizar investigaciones sobre cómo optimizar todos los procesos, y se comienzan a dar algunas soluciones que pasan por la separación del plano de control y de datos. Un ejemplo es la propuesta estandarizada por el IETF[8]: *ForCES, Forwarding and Control Element Separation*. Esta propuesta define la arquitectura y protocolos de comunicación entre el plano de control y el de datos, siendo así la predecesora de la propuesta que actualmente más fuerza está cogiendo: OpenFlow.

OpenFlow [9] es un estándar que define una arquitectura que también separa el plano de control y de datos, por lo cual esto no es una novedad como hemos apuntado ya en el párrafo anterior, pero su mayor aceptación y flexibilidad han hecho que sea esta la opción a tener en cuenta en cuanto a redes definidas por software se refiere. OpenFlow trata de dar una solución homogénea a cualquier arquitectura de red independientemente del fabricante, lo cual abre el candado que hoy en día los fabricantes imponen con sus soluciones propietarias. Ya existen implementaciones reales de switches compatibles con Openflow, de forma que estos equipos de red mantienen su funcionalidad en la capa de reenvío de paquetes pero el plano de control queda delegado en el/los controladores de Openflow.

Por otro lado, el nivel de abstracción que se propone es tal que, idealmente, no sería necesario tener tantos conocimientos a bajo de nivel de redes para poder gestionarlas y establecer políticas de uso de las mismas, ya que este es precisamente el trabajo que hacen los controladores de OpenFlow (y el de los proxys de estos controladores). Esto lleva a que los costes operacionales en clientes finales o consumidores de estas tecnologías sean menores, ya que el conocimiento experto sobre protocolos de red quedaría del lado de los que implementan las soluciones de redes definidas por software (SDN en adelante), y no de mano de quién las consume. Además, la escalabilidad es mucho mayor, se pueden introducir nuevos elementos de red sin que esto se traduzca en varias horas (o días) de trabajo de un operador de red especializado. Al tener todas las políticas de gestión de la red unificada en los controladores, también se evitan los fallos humanos a la hora de tener que configurar de forma homogénea toda la red pero modificando cada elemento de red por separado.

En temas de adaptación, la posibilidad de poder comunicarse con los controladores a través de, por ejemplo, interfaces REST (esto no todos lo implementan), hace que las propias aplicaciones puedan solicitar a la red directamente sus necesidades, de forma que el comportamiento de ésta sería totalmente adaptativo y flexible, convirtiéndose así en una red inteligente capaz de asignar los recursos de forma dinámica sin necesidad de intervención por parte de un operador de red. Esta característica es tan innovadora como peligrosa, ya que hay que poder establecer reglas que controlen estas peticiones y evitar que las aplicaciones, ya sea por

malfuncionamiento o por funcionamiento malintencionado, puedan modificar las reglas que se aplican para gestionar el tráfico pudiendo deteriorar o incluso dejar sin servicio al resto de equipos.

A nivel de seguridad [10], existen soluciones que añaden esta capa a distintos niveles de la arquitectura SDN: autenticación, control de aplicaciones, detección de patrones, detección de conflictos entre reglas... Aun así, y como ocurre en todos los sistemas, cada vez que se introducen nuevos elementos de control, inherentemente se abren más puertas a las posibles vulnerabilidades que pueden ser explotadas. Es por ello que, dado el impacto que puede llegar a tener el uso fraudulento de los controladores SDN, se deben implementar tantas medidas de seguridad como sean posibles, siempre que el impacto en la gestión de la red no sea relevante (si lo fuera, habría que buscar otras opciones de seguridad mejor implementadas) y que la calidad del servicio percibida por los usuarios y/o aplicaciones tampoco se vea afectada. Algunas investigaciones ponen de manifiesto que la seguridad puede ser el punto con menor madurez en las implementaciones de los controladores, de forma que cada uno responde de una forma diferente ante la llegada de paquetes con información errónea, este tipo de verificaciones debería estar estandarizado de alguna forma para que asegure la integridad y la fiabilidad de las redes que se gestionan con estos controladores.

Por otro lado, comienzan a surgir herramientas adicionales cuyo objetivo es facilitar aún más las labores de gestión de las SDNs. Así, se pueden ver nuevos sistemas operativos de red (NOS) orientados a las SDN o definición de nuevos lenguajes de programación para SDNs que ofrecen funcionalidades que ayudan y facilitan el trabajo a los desarrolladores de estas arquitecturas.

Como ejemplo de NOS, podemos considerar a los controladores de Openflow, que se detallarán más adelante, ya que éstos ofrecen un primer nivel de abstracción que permite gestionar de una forma más sencilla las diferentes redes. No obstante existen proyectos que dan un paso más y ofrecen otras capacidades y funcionalidades que también ayudan y abstraen de la gestión pesada, dura y tradicional de las redes de comunicaciones. Por ejemplo, el proyecto ONOS [3] sería un ejemplo de NOS a tener en cuenta. Más adelante se detallarán algunas de sus principales funcionalidades.

También han surgido otros proyectos, como lenguajes de programación para SDNs, los cuales tienen por objetivo crear interfaces de más alto nivel que ayuden a los desarrolladores de soluciones SDNs a implementarlas mediante lenguajes más intuitivos y con primitivas más sencillas y con mayor funcionalidad. Es el caso de Pyretic [11], un lenguaje de programación creado dentro de la familia de lenguajes de programación de Frenetic para SDNs. Este lenguaje de programación, basado en Python, permite a los operadores y desarrolladores de aplicaciones SDN la creación de nuevas aplicaciones modulares y con un alto nivel de abstracción.

Si hacemos referencia a las tendencias de búsqueda en Google sobre las arquitecturas SDN, vemos que están teniendo un gran interés a nivel mundial:

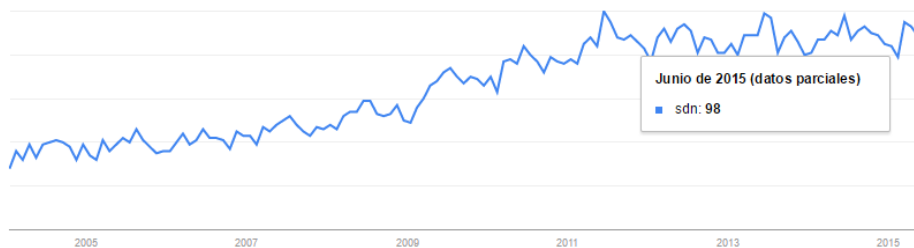


Fig. 1: Porcentaje interés de búsquedas web del término: SDN (en Google)

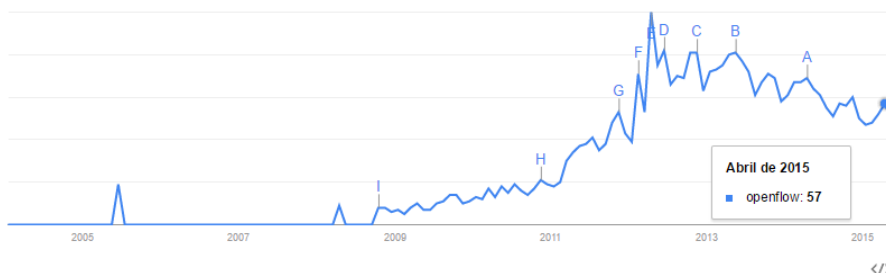


Fig. 2: Porcentaje interés de búsquedas web del término: Openflow (en Google)

Desde este punto de vista parece que también se puede inferir que el uso de estas tecnologías o el interés por ellas es cada vez mayor o al menos mantiene a día de hoy un gran interés.

Arquitecturas SDN

A continuación se detallan los principales componentes de una arquitectura de SDN basada en el estándar de Openflow, así como las *interfaces* de acceso a los *controladores SDN*.

Openflow

Openflow es un estándar abierto que permite el uso de las capacidades de los dispositivos de red que lo implementen sin necesidad de conocer el funcionamiento interno de éstos. De esta forma se pueden establecer reglas de gestión de flujos de red sin necesidad de utilizar las configuraciones propietarias del fabricante, lo que permite la interoperabilidad entre los dispositivos compatibles con Openflow independientemente del fabricante.

Este estándar ha evolucionado bastante desde sus principios en 2006 con la aparición de Ethane de manos del en aquél entonces estudiante de doctorado Martin Casado [12] hasta la última versión lanzada en diciembre de 2014, la 1.5[13], de mano de la Open Networking Foundation [14]. Actualmente Openflow ofrece características muy avanzadas, como es el control de diferentes espacios de red en diferentes tablas de flujos sincronizadas, posibilidades de filtrado muy granulares, por ejemplo, *flags* de TCP, uso de puertos de fibra óptica, etc.

La arquitectura que establece Openflow se basa en lo ya comentado anteriormente, la separación del plano de control y de datos. En los switches tradicionales estas dos funciones residen en cada uno de los dispositivos, mientras que con Openflow las reglas de

encaminamiento del tráfico (plano de control) se delegan en otro dispositivo centralizado, el controlador, quedando sólo el plano de datos en el switch.

De esta forma se centraliza la gestión de todo el tráfico de la red en el/los controladores Openflow, lo cual es una gran ventaja ya que no hay que ir conectándose a cada dispositivo para realizar las configuraciones pertinentes. Esta forma tradicional de configuración de la electrónica de red es muy tediosa y en escenarios de redes grandes y complicadas es difícil mantener la coherencia de todas las configuraciones. Con Openflow, al disponer de la gestión centralizada es más sencillo de mantener la configuración de todos los dispositivos, además de evitar errores de configuración, agilizar los cambios en las configuraciones de toda la red dotando a éstas de una mayor escalabilidad y fiabilidad.

El estándar de Openflow define los mensajes que se intercambian entre los dispositivos de red y los controladores a través de lo que se denomina en arquitecturas SDN *Southbound API*. Es a través de esta interfaz que, cuando llega un flujo de tráfico al dispositivo de red que no está definido en sus tablas de flujo, lo reenvía al controlador. Éste lo analiza y responde al dispositivo con una nueva entrada para la tabla de flujo que establece qué hacer con ese tipo de tráfico, de forma que desde ese momento el switch tratará a este flujo y a los posteriores con las mismas características de la misma forma, es decir, tal y como el controlador se lo acaba de indicar con esa nueva entrada.

En las tablas de flujos cada entrada tiene una serie de propiedades. Además del campo donde se establece el filtro para ese tipo de tráfico (*match field*), existe un campo de prioridad (toma de decisiones basado en prioridades), otro de *actions* donde se especifica el conjunto de acciones a realizar con ese flujo, otro de límite de tiempo (*timeout*) que establece el tiempo máximo que esa entrada es válida, y algunos otros que se pueden ver en detalle en la especificación [15].

Interfaces

Las interfaces en Openflow o en arquitecturas SDN son típicamente dos: la *Southbound API* introducida anteriormente y la *Northbound API*, la cual es ofrecida desde los controladores hacia las aplicaciones para que éstas puedan comunicarse con ellos. A continuación se ven un poco más en detalle.

Northbound API

Existe una gran variedad de controladores y cada uno ofrece sus propias interfaces para que las aplicaciones puedan comunicarse con ellos. Esta interfaz hacia el exterior es la llamada *Northbound API (NB API)*. La naturaleza de estas interfaces es heterogénea [16]: interfaces REST, RPC, OSGi, etc.

Gracias a estas interfaces se consigue la interacción entre aplicaciones y controladores, pudiendo conseguir configuraciones de red adaptadas a las peticiones de las aplicaciones. De cara a tener un mayor control sobre las peticiones de las aplicaciones, es en esta interfaz donde de alguna forma de debe controlar quién puede establecer reglas sobre qué tráfico. Para ello se podrían implementar mecanismos de control de acceso basado en roles de autenticación, por ejemplo. No obstante en este proyecto esta característica se va a implementar con el uso de la herramienta de Flowvisor, la cual se comenta en detalle más adelante.

Southbound API

El control de estas configuraciones se lleva a cabo en los controladores (conforman el plano de control). Como se ha comentado antes, son los controladores los que a través de la llamada *Southbound API (SB API)* envían las reglas sobre los flujos de tráfico a los dispositivos de red. El esquema general de estas arquitecturas con estas interfaces lo podemos ver en la siguiente imagen, obtenida de [16]:

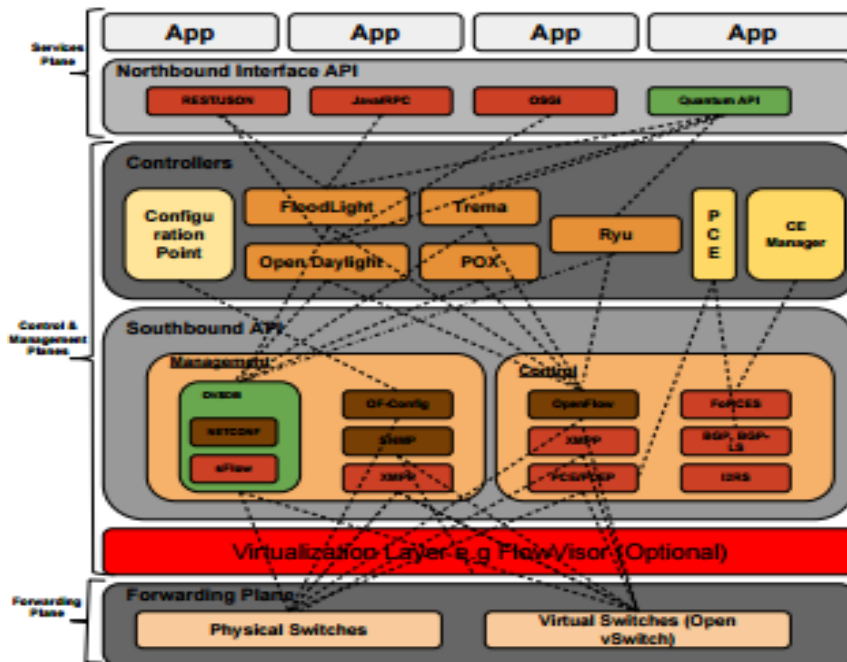


Fig. 3: Arquitectura SDN – Interfaces y componentes

Además, se pueden añadir capas intermedias que pueden implementarse a modo de *proxy* entre los controladores y los dispositivos de red, de forma que se podrían estar utilizando diferentes controladores, con NB APIs diferentes, para gestionar el mismo dispositivo. Un ejemplo de esta implementación es el ya citado anteriormente *Flowvisor*, a través del cual se puede dividir la red en *rodajas (slices)* y presentar estas vistas a los controladores que corresponda, obteniendo por consiguiente un particionado de la red. Estas características y otras que ofrece Flowvisor se exponen en detalle más adelante.

Finalmente, se tiene la red propiamente dicha, donde reside el plano de reenvío de paquetes, es decir, donde encontramos los switches físicos y/o virtuales que estamos gestionando desde los controladores.

Controladores

Existen una gran cantidad de soluciones open source que implementan este plano de control en medio de una arquitectura SDN. En el ámbito comercial son también muchas las soluciones que podemos encontrar y de mano de casi todos los grandes fabricantes de soluciones de redes: Cisco, Juniper, Brocade, Hp, etc. Se enumeran a continuación algunos de ellos [17]:

- Cisco: APIC, XNC
- Juniper: NorthStar Controller, Contrail (open)
- Brocade: Vyatta Controller
- HP: Virtual Application Networks (VAN) SDN Controller

En este proyecto se utilizará no obstante un controlador open source para la implementación de los escenarios y de su gestión. Entre los controladores open source más relevantes hoy día, podemos destacar:

- **Floodlight:**
 - Basado Java
 - NB API bajo servicios REST
- **POX:**
 - Versión de **NOX** (C++)
 - Implementado con Python
- **Beacon:**
 - Basado en Java
 - NB API bajo frameworks de OSGi y Spring
- **OpenDayLight (ODL):**
 - Basado en Java
 - NB APIs con OSGi y REST

De estos controladores el que más seguimiento tiene y evolucionado está a día de hoy es Floodlight, que ya es compatible con las versiones de OpenFlow 1.0 y 1.3, y en versión beta, también con la 1.2 y la 1.4. Por ello será éste el controlador que se utilice en las diferentes implementaciones de escenarios.

En la siguiente tabla, basada en [18], se pueden ver más características de cada uno de estos controladores:

	Beacon	Floodlight	NOX	POX	ODL
Soporte Openflow	OF v1.0	OF v1.0,v1.3 OF v1.2,v1.4 (beta)	OF v1.0	OF v1.0	OF v1.0,v1.3
Lenguaje de desarrollo	Java	Java	C++	Python	Java
REST API	No	Si	No	No	Si
GUI	Web	Web	Python+,QT4	Python+,QT4,Web	Web
Plataformas soportadas	Linux,Mac OS,Windows y Android	Linux, Mac OS,Windows	Linux	Linux, MacOS,Windows	Linux, MacOS, Windows
Soporte OpenStack	No	Si	No	No	Si
Multiprocesos	Si	Si	Si	No	Si
Código abierto	Si	Si	Si	Si	
Tiempo en el mercado (aprox)	5 años	3 años	7 años	2 años	1 año
Documentación	Buena	Buena	Media	Pobre	Media

Fig. 4: Controladores open source - características

La lista podría ser mucho más larga [19], pero se mencionan los controladores open source más utilizados a día de hoy.

Adicionalmente, existen otros proyectos algo más ambiciosos que los controladores de OpenFlow. Es el caso del proyecto ONOS [20], el cual se perfila como un sistema operativo de red que no solo ofrece una gestión centralizada y simplificada de la red, sino también ofrece características de mejora de rendimientos, escalabilidad y robustez. ONOS, en su definición de SB API por ejemplo, no sólo implementa Openflow, sino también otros protocolos, como puede ser NetFlow, de forma que es transparente de cara a la configuración de la red desde ONOS. Además, con el uso de los *intents*, se establecen las reglas de acceso a la gestión de los diferentes flujos de tráfico de la red. Sería algo parecido a lo que implementa Flowvisor, pero integrado en propio sistema operativo de red que conforma ONOS. En [3] se puede ver más en detalle las características generales de este proyecto.

Beacon

Este controlador [21][22] está implementado 100% en Java, y ha sido la base para la creación del controlador comentado anteriormente, Floodlight. Se creó en 2010 cuando aún no existía ningún otro controlador programado en este lenguaje, pero al ser un lenguaje independiente de la plataforma, ofrecer soporte para *multi-threading*, ofrecer buenos rendimientos y ser conocido por una gran parte de la comunidad de desarrolladores, fue elegido ante otros como C# o Python para implementar el controlador.

En la implementación de Beacon se utiliza la librería de Spring, la cual ofrece componentes que mejoran notablemente la programación del software, como es la *IoC (Inversion of Control)* y el *Web Framework*.

Beacon ofrece soporte con el estándar Openflow 1.0, y para interactuar con éste, Beacon utiliza la implementación de Java OpenFlowJ. Se pueden distinguir cuatro módulos principales (cada cual tiene su API) en este controlador:

- **Device Manager:** Encargado de registrar los switches vistos en la red así como algunas de sus características (direcciones Ethernet e IP vistas, última conexión, puertos...).
- **Topology:** Encargado de registrar los enlaces entre los switches, así como de recibir los eventos de nuevos enlaces o enlaces eliminados.
- **Routing:** Provee un mecanismo de *camino más corto (Shortest Path Layer)* para el enrutamiento entre los dispositivos de la red.
- **Web:** Proporciona la interfaz de usuario web.

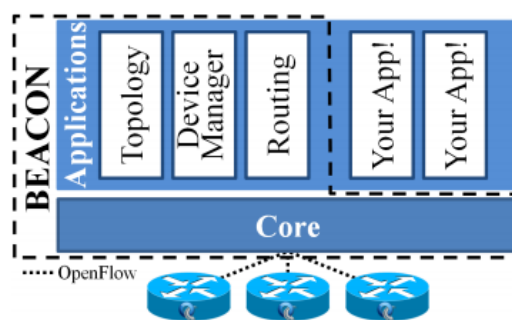


Fig. 5: Beacon overview [34]

En [22] se puede ver también un ejemplo de estudio de rendimiento de este controlador, el cual parece ser bastante bueno frente a los resultados obtenidos con otros controladores.

Nox/Pox

NOX es un controlador SDN implementado en C++ y fue donado por Nicira en 2008 a la comunidad. Es compatible con Openflow 1.0 y ofrece componentes de descubrimiento de topología de red o *learning switch* [23]. Ofrece soporte para los sistemas operativos Ubuntu 11.1 y 12.4 así como RedHat 6.

Por su parte POX es un hijo de NOX pero que ofrece un mejor entorno de desarrollo, ya que está creado con Python y ofrece soporte multiplataforma, ya que puede ser empaquetado y utilizado con *PyPy* [24] tanto en Linux, como en MAC OS o Windows. Al igual que NOX, ofrece soporte para la versión 1.0 del estándar de Openflow.

OpenDayLight

OpenDayLight (ODL) se presenta como un proyecto cuya intención es ir más allá de las implementaciones SDN con Openflow. Este proyecto nace en el seno de la Linux Foundation y es participado por un gran número de empresas y fabricantes de soluciones de red (tales como Cisco, Citrix, HP, Intel, Brocade, etc) [25].

Así, este proyecto además de ser compatible con Openflow, también lo es con otros lenguajes como VxLAN, pero la intención de ODL es que llegue a ser compatible también con la mayoría de los protocolos de los diferentes fabricantes, para poder controlar no sólo dispositivos compatibles con OpenFlow, sino también otros como pueden ser los que conforman las redes de acceso a las cabinas de almacenamiento (switchs de fibra por ejemplo).

La arquitectura base es similar a las del resto de controladores. El acceso a ODL (NB API) se implementa mediante interfaces REST que permiten interactuar con el controlador, al igual que ocurría con Floodlight. En la siguiente imagen se muestra esta arquitectura:

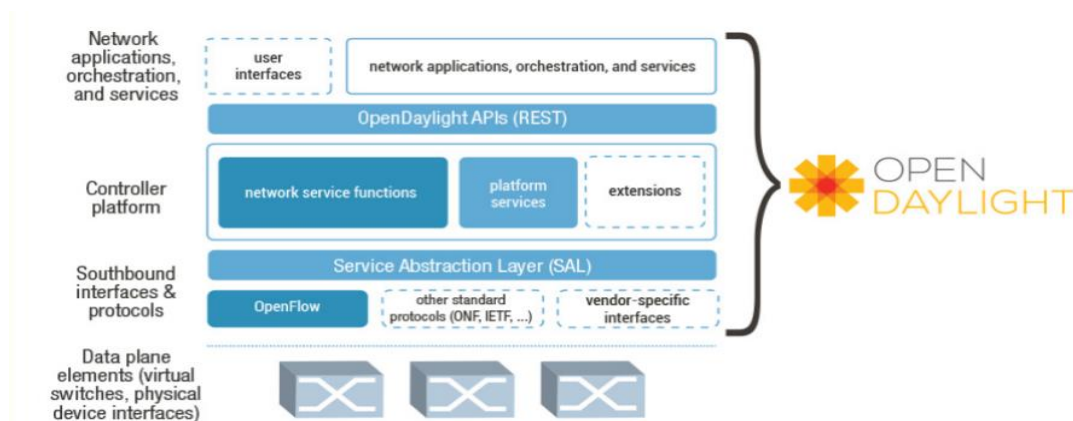


Fig. 6: OpenDayLight overview

Además de lo anterior, ODL también ofrece soporte para OpenStack de forma que se puede integrar en esta arquitectura facilitando así la gestión de la red en estos entornos.

Floodlight

Floodlight es un controlador SDN de propósito general que está desarrollado en Java bajo licencia de Apache. El soporte de esta herramienta es llevado a cabo por desarrolladores de la comunidad entre los que se encuentran ingenieros de Big Switch Networks. Con Floodlight se pueden conectar una gran variedad de dispositivos de red tanto físicos como virtuales ofreciendo a éstos un alto rendimiento en su gestión, de hecho, los productos comerciales de Big Switch Networks utilizan el framework de Floodlight en su software base. La última versión de este controlador es la 1.1, y fue liberada a mediados de Abril de 2015, fecha bastante reciente lo que corrobora que se trata de un proyecto vivo y con constante evolución.

Este controlador dispone de algunas características que le perfilan como uno de los controladores SDN open source con más fuerza. Una de ellas es la compatibilidad a día de hoy con casi todas las versiones del estándar Openflow, ya que soporta las versiones 1.0 y 1.3.x, y en versión beta también lo hace con las versiones 1.2 y 1.4. En su implementación cabe destacar también los módulos implementados y aplicaciones dentro de Floodlight accesibles a través de API REST, como son:

- **Listas de control de acceso (ACL's):** De forma proactiva se configuran reglas de *allow* y/o *deny* para diferentes tráficos con cierta dirección ip origen o destino. Las reglas de flujo que se crean desde este módulo tienen son entradas estáticas en las tablas de flujos y cuyo nombre tendrá la forma:
 - ACLRule_[Rule Id]_[Switch_DPID]Estas entradas pueden ser controladas a través del módulo que se ve algo más bajo llamado *Static Flow Pusher*.
- **Firewall:** Este módulo ofrece opciones de filtrado de tráfico, de forma que una vez habilitado, por defecto ningún flujo está permitido, y habrá que ir añadiendo reglas de flujos que serán los que se permitan a través del controlador. Esta forma de trabajo es más bien reactiva, al contrario que con las ACL's.
- **Static Flow Pusher:** Esta funcionalidad nos permite crear reglas que serán posteriormente enviadas a los dispositivos de red correspondientes conformando así sus tablas de flujos. Utilizando esta herramienta se consigue dinamizar la configuración de las redes desde las propias aplicaciones, de manera que, de una forma controlada, las aplicaciones podrán modificar el comportamiento de la red en función de sus necesidades. Por ejemplo, para configurar una nueva regla, utilizando la utilidad de peticiones POST *curl*, esta sería la notación a seguir:
 - `curl -d '{"switch": "00:00:00:00:00:00:00:01", "name": "flow-mod-1", "cookie": "0", "priority": "32768", "in_port": "1", "active": "true", "actions": "output=2"}' http://<controller_ip>:8080/wm/staticflowpusher/json`
 - En este ejemplo se establece que los flujos en el switch con DPID=:01 que entren por el puerto 1, salgan por el puerto 2.
 - Las opciones de definición de flujos (en el ejemplo, *in_port=1*) pueden anidarse separando por comas cada uno de los filtros.
 - Las acciones a realizar (en el ejemplo solo hay una: *output=2*), podrían ser un conjunto de acciones (*action-set*), tal y como se define en el estándar de Openflow. La forma de tratar estas acciones es diferente en la versión 1.0 Openflow y en las posteriores, ya que en éstas las diferentes acciones van actualizando la tabla de acciones (a medida que se van evaluando las diferentes tablas de flujos) que finalmente se llevarán a cabo con el flujo definido, y en la versión 1.0 se ejecutan directamente (sólo hay una tabla de flujos).

- Existen diferentes tipos de acciones, como pueden ser:
 - Descartar el paquete
 - Modificar parámetros de nivel 3: decrementar TTL, direcciones origen/destino, dscp, etc.
 - Push/Pop de tags MPLS o de VLAN
 - Reenvío de flujo al controlador

En la resolución de los escenarios propuestos más adelante, es esta funcionalidad la que se utiliza para la configuración de las reglas de cada controlador, de manera que se podrán ver más ejemplos de uso de la misma.

- **Virtual Network Filter:** También conocida como *Virtual Switch*, esta aplicación permite la creación de redes virtuales de nivel 2 que puede utilizar como aplicación *standalone* o bien como *back-end* de OpenStack Quantum, utilizando el plug-in de Quantum, que ofrece modelos de redes como servicio (NaaS) a través de un API REST que es implementada en Floodlight.
- **Circuit-Pusher:** Esta aplicación utiliza el API REST de Floodlight para poder crear circuitos bidireccionales entre dos dispositivos dados sus direcciones IPs. La aplicación se encarga de configurar las reglas de flujo en el controlador necesarias para que esta comunicación bidireccional se lleve a cabo con éxito.
- **Forwarding:** Este módulo es utilizado por defecto en los controladores Floodlight. Se encarga de insertar las reglas de flujos necesarias para comunicar un origen y un destino una vez han sido descubiertos mediante los mecanismos convencionales. Esta forma de funcionar puede dar lugar a problemas de cara a la memoria ocupada por las tablas de flujos en los switches, y para evitarlo, en la configuración del módulo en Floodlight (archivo *floodlightdefault.properties*) se pueden limitar reduciendo el criterio de asociación de los flujos, para por ejemplo utilizar solo los campos de vlan, mac, ip y transporte:
 - `net.floodlightcontroller.forwarding.Forwarding.match=vlan, mac, ip, transport`
 También es posible deshabilitar este módulo eliminando la línea que hace que se cargue al iniciar el controlador, sería la línea:
 - `net.floodlightcontroller.forwarding.Forwarding`
- **Load Balancer:** Este módulo ofrece servicios de balanceo de carga para flujos icmp, tcp y udp. Es un módulo casi de ejemplo ya que es bastante limitado en cuanto a los servicios que ofrece. Es accesible a través de API REST y se ha creado de cara a ofrecer servicio de *Load-balancer-as-a-Service (LBaaS)* en el ámbito de OpenStack Quantum.

De forma general, la arquitectura de Floodlight con todas sus aplicaciones y módulo quedaría como sigue:

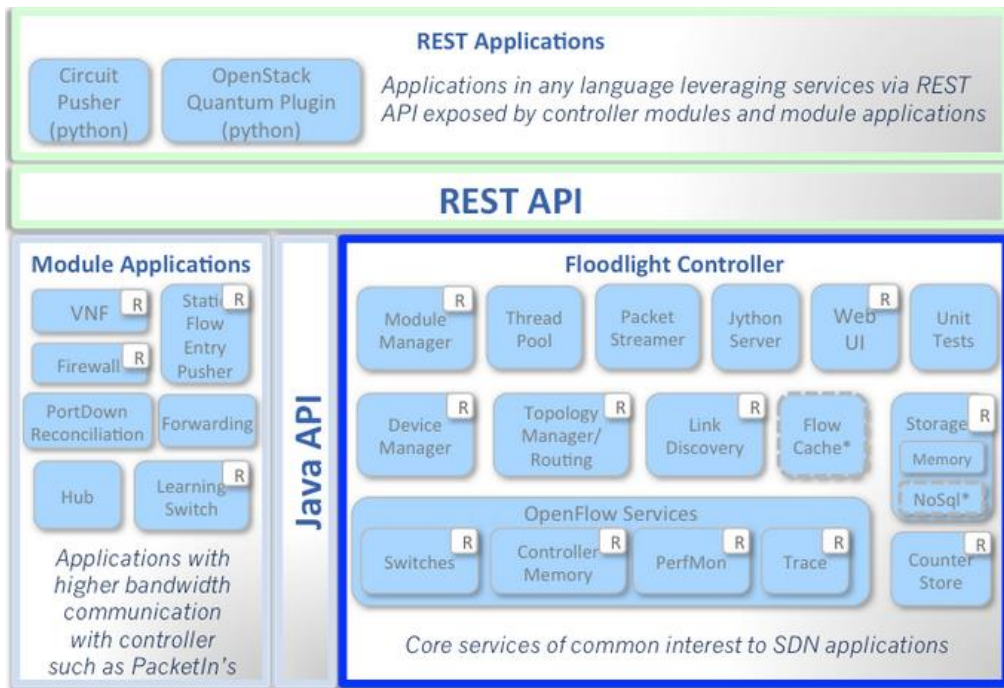


Fig. 7: Arquitectura Floodlight

Para ver más en detalle todas las opciones de todas estas aplicaciones, se puede consultar la web oficial del proyecto de Floodlight en [27]. Además, es de recomendada lectura también lo relacionado con la integración de Floodlight con OpenStack, lo cual no se verá más detalle en esta memoria por estar fuera del ámbito de este trabajo.

Floodlight también implementa una interfaz de usuario web a través de la cual se pueden ver los elementos de red que se han conectado al controlador, los puertos y los hosts vistos en esos puertos, así como las reglas de flujo que se han configurado en cada uno de los switches.

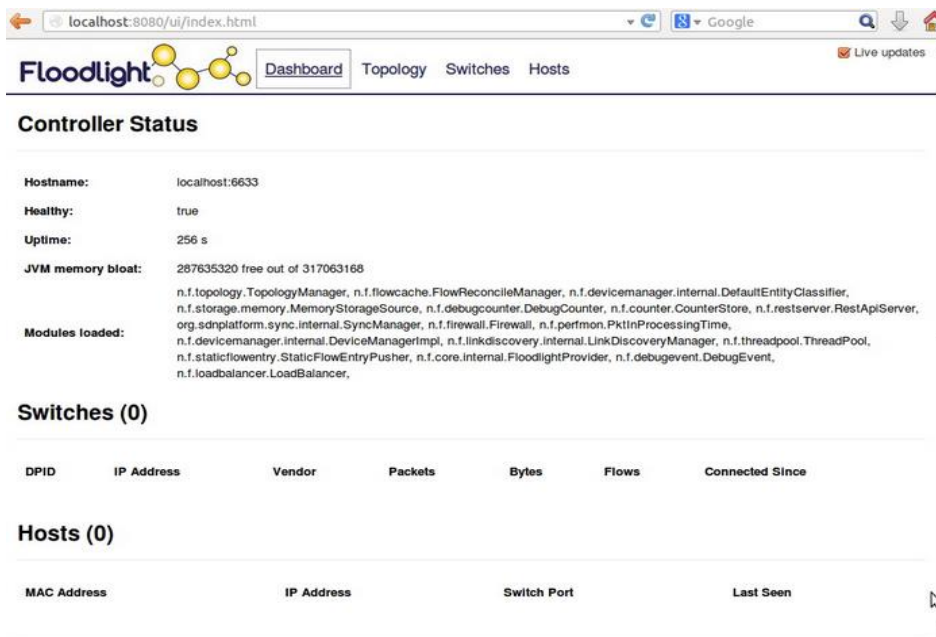


Fig. 8: Floodlight web UI - Dashboard

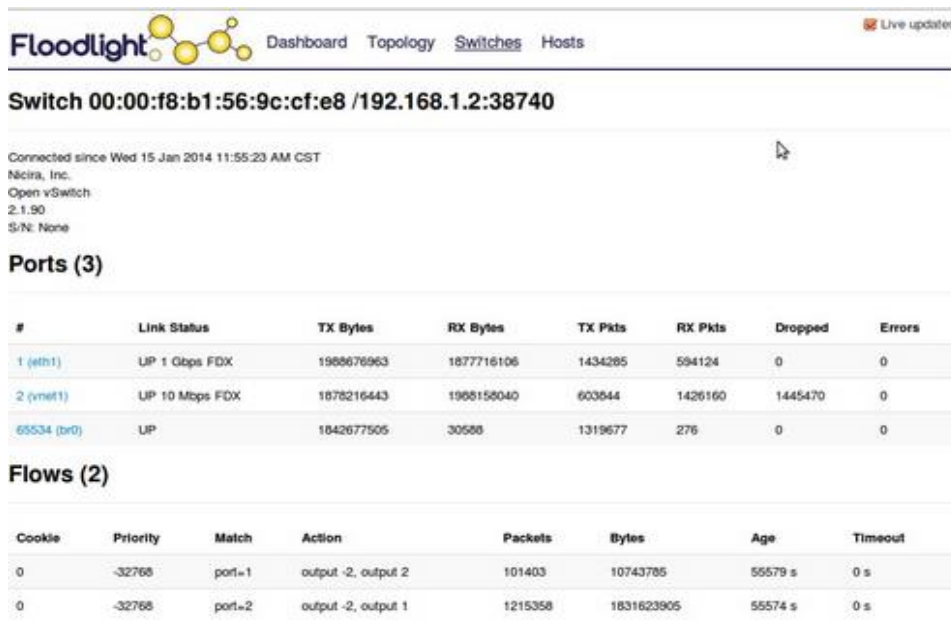


Fig. 9: Floodlight web UI - Detalle de un switch

Otra vista interesante es la que muestra la topología de red que el controlador Floodlight está viendo y sobre la cual podrá realizar acciones de monitorización y/o configuración:

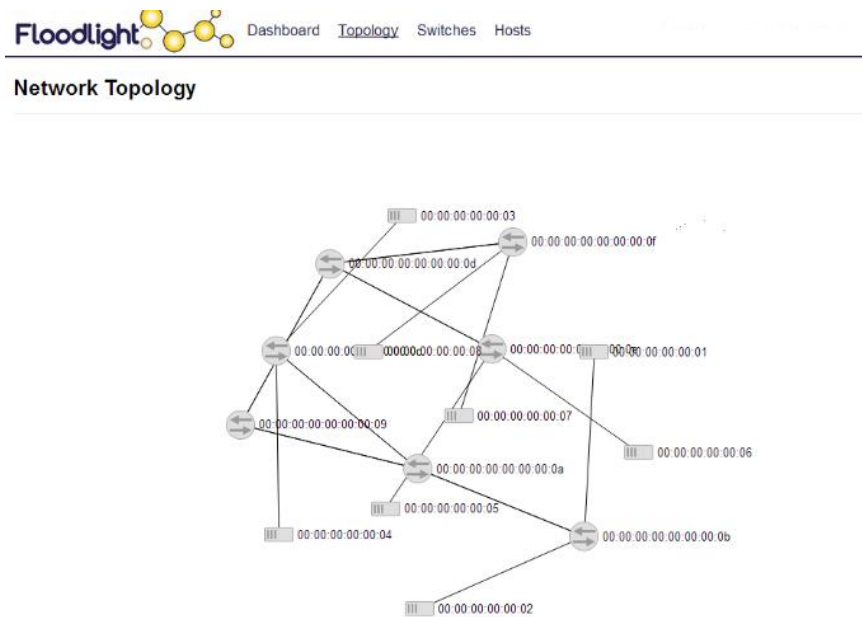


Fig. 10: Floodlight web UI - Topología de red

Instalación

Floodlight puede ser instalado en sistemas Linux o Mac, pero en este proyecto sólo se hablará de la versión de Linux. Como sistema Linux soportado se indica que debe ser Ubuntu 10.04 o posterior, y se deben tener instalados el software de *ant* y el *jdk* de java. Opcionalmente, para utilizar Floodlight desde el framework de desarrollo de Eclipse, este software también se puede instalar. Todo esto, con el siguiente comando quedaría instalado:

```
sudo apt-get install build-essential default-jdk ant python-dev eclipse
```

A continuación, simplemente hay que descargar el Proyecto de Floodlight, compilarlo y crear un directorio donde se creará por defecto la base de datos de Floodlight. Los pasos a seguir serían los siguientes:

- `git clone git://github.com/floodlight/floodlight.git`
- `cd floodlight`
- `ant`

- `sudo mkdir /var/lib/floodlight`
- `sudo chmod 777 /var/lib/floodlight`

A continuación, para iniciar el controlador, habría que ejecutar el comando:

- `java -jar target/floodlight.jar`

Existe un fichero de configuración de propiedades de Floodlight que cabe mencionar. Se encuentra, dentro del directorio raíz de Floodlight, en la siguiente ruta:

- `src/main/resources/floodlightdefault.properties`

En este fichero se pueden habilitar o deshabilitar los módulos que cargará el controlador al iniciar, además de poder configurar algunos de los puertos que utiliza éste (como el del API REST, el puerto de escucha para Openflow...). Por ejemplo, en casi todos los escenarios propuestos más adelante, existen un módulo que por defecto está incluido en este fichero pero que se ha quitado, es el módulo de Forwarding, cuyo cometido es que el controlador configure los flujos necesarios para conectar todos los equipos conectados en todos los switches, tal y como ocurriría con una red de switches sin controlador Openflow mediante los mecanismos de descubrimiento y convergencia convencionales. Decir que esta configuración de flujos se realiza a medida que va apareciendo tráfico a través de los switches conectados al controlador.

Virtualización

Flowvisor

Adicionalmente a la arquitectura estándar de las SDN vista hasta ahora, existe la posibilidad de utilizar herramientas intermedias que aumentan el control sobre el tráfico de la red. Es el caso de Flowvisor, una herramienta que ofrece la posibilidad de realizar un particionado (virtualización) de la red para así presentársela a los diferentes controladores con los permisos correspondientes. La versión actual de esta herramienta es la 1.4, pero no ha evolucionado desde Agosto de 2013, de manera que sólo ofrece compatibilidad con la versión 1.0 de OpenFlow.

El particionamiento se realiza en base a los diferentes valores de los paquetes recibidos en Flowvisor. Además, como utiliza Openflow, Flowvisor puede estar conectado a cualquier controlador, de la naturaleza que sea, siempre que éste implemente, al menos, el estándar de Openflow 1.0. De esta manera, cada controlador de SDN tendrá una vista parcial (o total si así se configura) de la red, y podrá gestionar lo definido en la partición que se le presenta. Los parámetros para definir cada partición, llamada *flowspace*, pueden ser los siguientes:

Identificador	Descripción	Ejemplos
in_port	Puerto físico de entrada	in_port=1
dl_vlan	Tag de VLAN según la IEEE 802.1q. Un valor de 0xffff hace referencia a los paquetes sin tag de vlan	dl_vlan=45
dl_src	Dirección MAC origen	dl_src=00:00:00:00:00:01
dl_dst	Dirección MAC destino	dl_dst=00:00:00:00:00:02
dl_type	Campo Tipo de protocolo en Ethernet	dl_type=0x0806 (tráfico ARP)
nw_src	Dirección IPv4 origen. Puede ser de la forma ip/máscara, especificando la máscara con notación CIDR	nw_src=10.0.0.1 nw_src=10.0.0.0/8
nw_dst	Dirección IPv4 destino. Puede ser de la forma ip/máscara, especificando la máscara con notación CIDR	nw_dst=11.0.1.1 nw_dst=11.1.2.0/24
nw_proto	Tipo de protocolo IP	nw_proto=6 (para TCP)
nw_tos	Identifica el campo tos/dscp de la cabecera de IPv4. Valor decimal entre 0 y 255.	nw_tos=2
tp_src	Puerto de origen del nivel de transporte. Valor decimal entre 0 y 65535	tp_src=80
tp_dst	Puerto destino del nivel de transporte. Valor decimal entre 0 y 65535	tp_dst=1433

Tabla 1: Identificadores de los flowspaces en Flowvisor

Con la lista de identificadores anterior se puede realizar cualquier combinación que nos convenga para definir los diferentes espacio de red que se van a presentar a los controladores. Se puede observar que los identificadores coinciden con los del estándar de Openflow 1.0, pero no con los definidos a partir de la versión 1.2, ya que como hemos dicho antes, Flowvisor solo ofrece soporte para la primera versión.

Por otro lado, existen algunas dependencias entre algunos parámetros, las cuales vienen definidas por el propio estándar de Openflow [15]. A medida que Openflow ha ido evolucionando estas dependencias también, y hay que tenerlas en cuenta de cara a definir los flowspaces (que en definitiva se traducirán en las reglas de flujos que se escribirán en la tabla de flujos de los dispositivos de red) y también las reglas de los controladores. Para una revisión detallada de todas las dependencias de cada identificador de cada versión de Openflow (hasta la 1.4), se puede visitar la web [26].

En relación a lo anterior cabe añadir que los parámetros no utilizados en la definición de los flowspaces utilizarán los llamados *wildcards*, de forma que en algunos casos esto significará que la omisión del parámetro significa que todos los valores para ese identificador estarán dentro del flowspace. En la web anterior también se pueden ver detalladamente estas correspondencias.

Cada una de las particiones definidas anteriormente pueden presentarse a uno o varios controladores SDN, de forma que estos recibirán los mensajes Openflow que coincidan con esos filtros definidos en el/los flowspaces. Para los controladores, así como para los dispositivos de

red, la existencia de Flowvisor en medio es totalmente transparente. De igual forma, las reglas de flujos que los controladores envíen a los dispositivos de red, serán filtradas y formateadas por Flowvisor, de forma que se asegura que las reglas definidas por cada controlador solo afectarán a los flujos pertenecientes al/los flowspaces definidos. A cada conexión con cada controlador Flowvisor lo llama *slice*. Para asignar los flowspaces a las slices, en la definición de éstos se agregan qué slices tendrán acceso a ese conjunto de flujos, y además se podrá definir diferentes niveles de permisos a cada slice.

De esta forma, un mismo flowspace puede estar asignado a la slice_1 (que conecta con el controlador_1) con permiso de escritura (puede hacer peticiones de datos con Openflow así como escribir reglas de flujos) y a la vez estar asignado a otra slice_2 (que conecta con el controlador_2), la cual tendrá, por ejemplo, solo permisos de lectura, es decir, si este controlador_2 envía alguna regla de flujo para que sea establecida en algún dispositivo, Flowvisor la interceptará y no dejará que se realice la operación. Además de los permisos de lectura y escritura, existe otro nivel de permiso, el de poder delegar la gestión de ese flowspace en otro controlador.

Para interactuar con Flowvisor existen dos comandos importantes: `fvconfig` y `fvctl`. El primero nos permite cargar, crear o eliminar datos en la base de datos de Flowvisor, como pueden ser los ficheros de configuración inicial de la herramienta. Con `fvctl` se dispone de las opciones de consulta y manipulación de los objetos de Flowvisor (slices y flowspaces), así como de opciones de salvado de la configuración.

Adicionalmente, Flowvisor puede ser configurado directamente con ficheros de configuración, xml o json, dependiendo de cómo se haya configurado en la instalación, aunque en primera instancia se recomienda el uso de los comandos comentados en el párrafo anterior para configurar y manipular ese software.

Se van a comentar más adelante algunos de los principales comandos a utilizar con Flowvisor, pero es de recomendada lectura el manual de uso de estos comandos (en Linux, `man fvctl` o `man fvconfig`).

Instalación

La instalación de Flowvisor es sencilla y se puede realizar de diferentes formas. Se va a detallar la utilizada en este proyecto, aunque en la web oficial del proyecto se ofrecen más opciones. En primer lugar vamos a indicar que se deben tener instalados, igual que con Floodlight, el software de java y ant (el comando sería el mismo que se indicó para el controlador Floodlight). Una vez preparado el software, Flowvisor necesita de la existencia de un usuario en el sistema que sea el que ejecute esta herramienta. Para ello, se crea el usuario llamado flowvisor con la contraseña que sea (hay que recordarla, ya que luego nos la pedirá durante la instalación). Para crear el usuario, se puede utilizar el comando:

- `adduser flowvisor`
- Rellenar los campos que nos va solicitando (se pueden dejar los valores por defecto)

El siguiente paso es descargar el proyecto de *github*, se accede al directorio de Flowvisor y se instala:

- `git clone git://github.com/opennetworkinglab/Flowvisor`
- `cd Flowvisor`

- make
- make install
- Aparecerán algunas preguntas (como usuario de Flowvisor, directorio de instalación, etc), se dejan todas las opciones por defecto pulsando *intro* en todas ellas.
- Si nos fijamos durante los logs de la instalación, veremos como se crea un enlace simbólico para el comando de fvctl a la implementación con json, fvctl-json.

A continuación vamos a generar un fichero básico de configuración de Flowvisor, el cual sólo contendrá los parámetros de inicialización del mismo y una slice de administración, cuyas credenciales se utilizarán con para después configurar todos los flowspaces y slices que necesitemos. Para ello, en primer lugar usaremos el comando fvconfig y generaremos ese fichero básico de inicio:

- Para usar fvconfig, Flowvisor debe estar parado.
- fvconfig generate /etc/flowvisor/config.json
 - Con este comando se genera un fichero llamado config.json en la ruta que pongamos (típicamente, /etc/flowvisor) con la configuración básica. Si vemos su contenido, tenemos:

```
{
  "switchs": [],
  "flowvisor": [
    {
      "api_webserver_port": 8080,
      "db_version": 2,
      "host": "localhost",
      "log_ident": "flowvisor",
      "checkpointing": false,
      "listen_port": 6633,
      "logging": "NOTE",
      "run_topology_server": false,
      "log_facility": "LOG_LOCAL7",
      "version": "flowvisor-1.4.0",
      "config_name": "default",
      "api_jetty_webserver_port": 8081,
      "default_flood_perm": "fvadmin",
      "track_flows": false,
      "stats_desc_hack": false
    }
  ],
  "FlowSpaceRule": [],
  "Slice": [
    {
      "contact_email": "fvadmin@localhost",
      "admin_status": true,
      "creator": "fvadmin",
      "passwd_salt": "906347476",
      "drop_policy": "exact",
      "config_name": "default",
      "max_flow_rules": -1,
      "name": "fvadmin",
      "controller_port": 0,
      "controller_hostname": "none",
      "flowmap_type": "federated",
      "passwd_crypt": "c4920e18bad7d0fc9feee0282f9742d4",
      "lldp_spam": true
    }
  ]
}
```

```
}  
]  
}
```

Como vemos, en este fichero se ve bastante información, como los puertos que se utilizan, la versión de Flowvisor, el usuario administrador (fvadmin), el nivel de *logging* con el que se ejecutará Flowvisor, etc.

Una vez instalado se podría ya ejecutar, no obstante se van a comentar algunos detalles y/o consejos a tener en cuenta antes de ejecutar Flowvisor:

- Creación de fichero de password para usuario de fvctl (típicamente, usuario fvadmin)
 - De cara a no tener que poner la password del usuario con permisos para configurar Flowvisor, que en el ejemplo anterior es fvadmin, se puede crear un fichero con estas credenciales, de manera que pasando como parámetro este fichero al comando fvctl, no es necesario escribirla en la consola de comandos. El contenido del fichero sería este solo debe contener en texto plano la password, y el comando de fvctl utilizando este fichero sería (suponiendo que el fichero sea /etc/flowvisor.passwd):

i. `fvctl -f /etc/flowvisor.passwd`

De cara a usar scripts para la ejecución de comandos con fvctl, es útil, incluso necesario, disponer de este fichero con la password. Por supuesto, ese fichero debe protegerse con los permisos de acceso correspondiente.

- Bug de parseo de identificador nw_dst
 - Flowvisor cuando tiene algún flowspace que utiliza el identificador de nw_dst o nw_src, donde se indica el valor en forma de dirección IPv4 y opcionalmente se añade la máscara de red en notación CIDR, este valor lo convierte en un número entero, el cual puede ser mayor de lo permitido por el objeto Long de java. En el código de Flowvisor, en concreto en la clase FlowSpaceImpl.java (se encuentra en src/org/flowvisor/config) se hace un parseo de este identificado a Long, pero por lo indicado anteriormente, debería estar parseado a Double (igual que nw_src, el cual sí está parseado a Double), por lo que se recomienda cambiar esta línea del código java (línea 865) y recompilar (con *ant* por ejemplo) el código de Flovisor. Este bug fue reportado en 2013, incluso se solicitó su inclusión en el código, pero después del lanzamiento de la última versión, y no ha sido añadido puesto que no ha habido versiones posteriores. Se puede ver este reporte en <https://github.com/opennetworkinglab/flowvisor/pull/269>:

```
861         ps.setInt(9, ((Double) row.get(NWSRC)).intValue());  
862         if (row.get(NWDST) == null)  
863             ps.setNull(10, Types.INTEGER);  
864         else  
865             ps.setInt(10, ((Long) row.get(NWDST)).intValue());  
866         if (row.get(NWPROTO) == null)  
867             ps.setNull(11, Types.SMALLINT);
```

Fig. 11: FlowSpaceImpl.java - Bug Double NW_DST

- Revisión de los puertos utilizados por Flowvisor

- Flowvisor utilizar por defecto algunos puertos, como el 8080 o 6633 que es donde escucha el tráfico Openflow (otra marca de que solo es compatible con Openflow 1.0 y que esta herramienta no está siendo actualizada, ya que desde Julio de 2013 el puerto asignado por el IANA para Openflow es el 6653, y es el que por defecto se usa con versiones posteriores a la 1.0 de Openflow). Si hay otras aplicaciones que utilicen esos puertos, se deberían modificar. En el caso de Flowvisor, se pueden modificar en el fichero que se utilice (o en la configuración cargada en la base de datos con fvconfig) para iniciar Flowvisor (en el fichero *config.json* visto anteriormente, se ven claramente estos parámetros).

Una vez revisados los puntos anteriores y con Flowvisor compilado, pasamos a ejecutarlo, para ello basta con ejecutar el comando:

- `sudo -u flowvisor flowvisor /etc/flowvisor/config.json`

Según el nivel de *logging* configurado (NOTE, DEBUG, INFO, etc), se verán más o menos trazas de lo que está haciendo Flowvisor. Además, es conveniente comprobar que efectivamente los puertos configurados son los que están esperando conexiones, para ello, se puede ejecutar el comando siguiente (o utilizar cualquier otra herramienta que nos muestre los puertos TCP que están escuchando en ese instante):

- `netstat -nctl`

En este punto, ya sería posible conectar los dispositivos de red correspondientes a Flowvisor. En estos dispositivos simplemente habría que configurar que el controlador Openflow está en la dirección IP de la máquina de Flowvisor y en el puerto 6653 (o el que se haya configurado). No obstante, hasta que no se configuren las slices y flowspaces, los controladores SDN no podrán ver ningún dispositivo, por ello vamos a detallar cómo hacerlo.

Slices

Para la creación de las slices, se deben conocer qué controladores SDN van a gestionar el tráfico de nuestra red y en qué dirección IP/puerto están disponibles. Con esta información, y haciendo uso del comando fvctl:

- `fvctl -f /etc/flowvisor.passwd add-slice sliceName tcp:10.2.3.4:6653 slice@mail.com`

Si se intentara crear un slice con un nombre ya existente, nos avisaría con un mensaje de error, por lo que cada slice debe tener un nombre diferente, ya que es este parámetro el que se utilizará a la hora de dar permisos a cada una de ellas en cada flowspace.

Flowspaces

Para configurar los flowspaces se utilizará también el comando de fvctl y será necesario conocer en detalle qué queremos configurar, es decir, conocer qué parámetros necesitamos filtrar para generar los flowspaces deseados. La descripción genérica de uso de este comando para crear flowspaces es la siguiente:

- `fvctl add-flowspace [options] <flowspace-name> <dpid> <priority> <match> <slice-perm>`

Donde *flowspace-name* es el identificador del flowspace (debe ser único), el *dpid* es el identificador del switch que presentaremos al/los controlador/es, *priority* es la prioridad de esta regla frente a otras con filtros similares (es un valor entero, y a mayor valor, mayor prioridad), *match* se refiere a los filtros que vamos a establecer y en *slice-perm* estableceremos a qué slices le presentamos el flowspace y con qué permisos (7=DELEGATE,4=WRITE,2=READ).

En el siguiente ejemplo, se establece un flowspace llamado `fv_fs_http1` con prioridad igual a 100 que filtra el tráfico del switch con identificador `:::01`, que entra por el puerto 2 de este switch, la ip origen es 10.0.0.1, la ip destino es 10.0.0.6, el tipo de protocolo es IP (0x0800), el protocolo de transporte es TCP y el puerto destino es el 80. Este flowspace se presentará a la slice con nombre `sliceHTTP1` con permisos de escritura, y además, se fuerza a que de las dos colas QoS disponibles (0 y 1) este tráfico sea encolado siempre por la cola QoS 0 (la configuración de las colas QoS se debe realizar localmente en cada switch que se necesite, Openflow no abarca esta configuración, sólo hace uso de ella):

- ```
fvctl -f /etc/flowvisor.passwd add-flowspace fv_fs_http1
00:00:00:00:00:00:00:01 100
in_port=2,nw_src=10.0.0.1,nw_dst=10.0.0.6,dl_type=0x0800,nw_prot
o=6,tp_dst=80 sliceHTTP=4 -q 0,1 -f 0
```

Se puede comprobar que la granularidad de definición de flowspaces es muy alta. Como resultado de la creación del flowspace anterior, el controlador configurado en la llamada `sliceHTTP1` vería que se acaba de conectar un switch con un solo puerto. Si en este controlador se establecen las reglas que sean (como las que se comentaron anteriormente con Floodlight), Flowvisor las interceptará y añadirá los filtros configurados en el flowspace antes de enviárselas al switch para que las incluya en su tabla de flujos. Si se intentaran establecer reglas fuera del ámbito de este flowspace desde el controlador, Flowvisor lo impediría.

De esta forma se consigue aportar aún más valor a la arquitectura típica de las SDN, pudiendo diseccionar la red de muy diferentes formas, no sólo basándonos en ip's origen/destino o tags de MPLS o VLAN. Así, con Flowvisor se pueden crear escenarios donde se delega el control del tráfico con ciertas características en uno o varios controladores (por ejemplo, para pruebas de nuevos protocolos o desarrollos), mientras que el resto del tráfico (por ejemplo, el tráfico de producción) no se vería afectado aun cuando se comentan errores o existan fallos en las configuraciones establecidas en esos controladores. En los escenarios propuestos, se ponen de manifiesto todas estas características que ofrece Flowvisor.

Algunos ejemplos gráficos sobre el particionado de la red que ofrece Flowvisor queda bien definido con la siguiente imagen, donde con una misma arquitectura de red física se pueden presentar variadas porciones de la red en función de los parámetros de las cabeceras de nivel 1 a nivel 4:

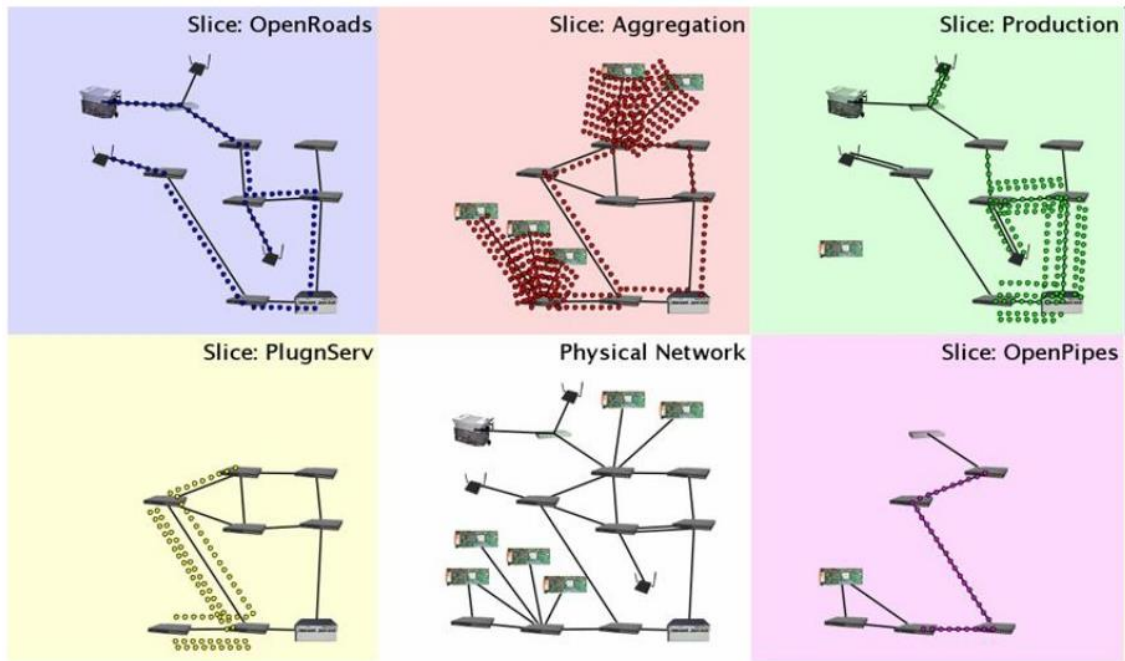


Fig. 12: Particiones Flowvisor

Por último, simplemente comentar que tras contactar con los desarrolladores de ON.lab, desarrolladores principales de Flowvisor, confirmaron que el soporte sobre esta herramienta ya no existe y que no hay intención, al menos de momento, de retomar el proyecto. Era algo previsible viendo la nula actividad que tiene el proyecto desde 2013, lo cual también se refleja en las tendencias de búsqueda en la web del término Flowvisor (en Google):



## OVX: OpenVirteX

Este software, desarrollado como Flowvisor en ON.lab, se perfila como una herramienta complementaria a Flowvisor que ofrece la posibilidad de creación de diferentes redes virtuales sobre la misma red física, de forma que cada entorno de red virtual puede utilizar cualesquiera de los dispositivos físicos que estén disponibles para conformar es nueva red virtual, que es la que en definitiva se le presentaría al/a los controladores SDN. De esta forma, se consigue también un particionado de la red, pero visto con un prisma diferente.

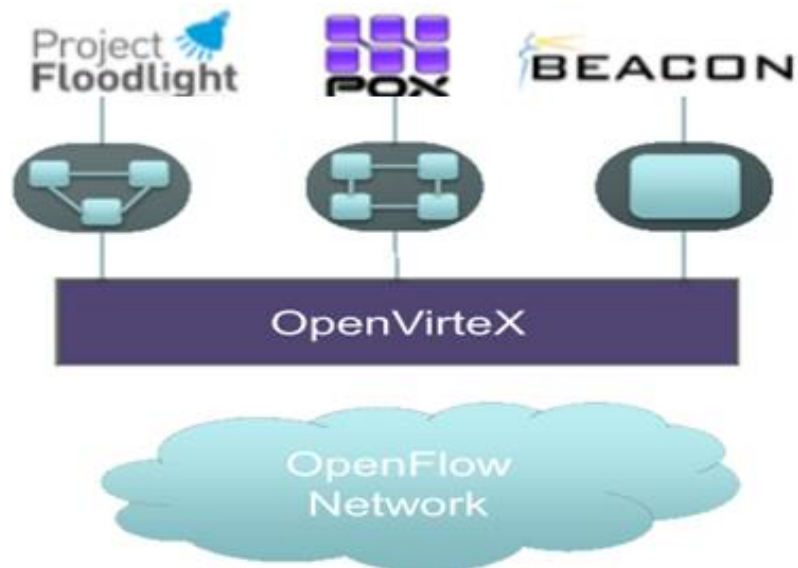


Fig. 13: OVX – Arquitectura conceptual

Con el uso de las diferentes redes virtuales, OVX consigue el aislamiento del tráfico en cada una de ellas. Estas redes virtuales tienen un direccionamiento virtual, y OVX se encarga de conservar el mapeo de las direcciones virtuales con las físicas, de forma que así asegura el aislamiento entre las diferentes redes presentadas a los diferentes NOS (controladores Openflow, proxys Openflow como Flowvisor, ONOS, etc.). Así, y aunque no se ha implementado en este trabajo, se podría utilizar OVX en conjunto con Flowvisor y Floodlight, consiguiendo así un nivel más de abstracción, o lo que es lo mismo, de virtualización, pero con la posibilidad de hacer uso de las ventajas que ofrece OVX.

Además de la creación de los espacios virtuales de red, OVX ofrece bastantes más funcionalidades interesantes. Por ejemplo, con OVX ofrece la gestión ante posibles caídas de enlaces, de forma que el tráfico se enviaría por otros caminos alternativos, pero consiguiendo que no haya pérdida del servicio. Habría que ver cómo combinar este tipo de funcionalidad con Flowvisor y ver si afecta algo en la configuración que deba tener éste, aunque se supone que OVX es transparente para lo que haya por encima. OVX también ofrece soporte para OpenStack con el uso de su plugin de *Neutron* [28], aunque aún no está incluido en el código fuente de OpenStack.



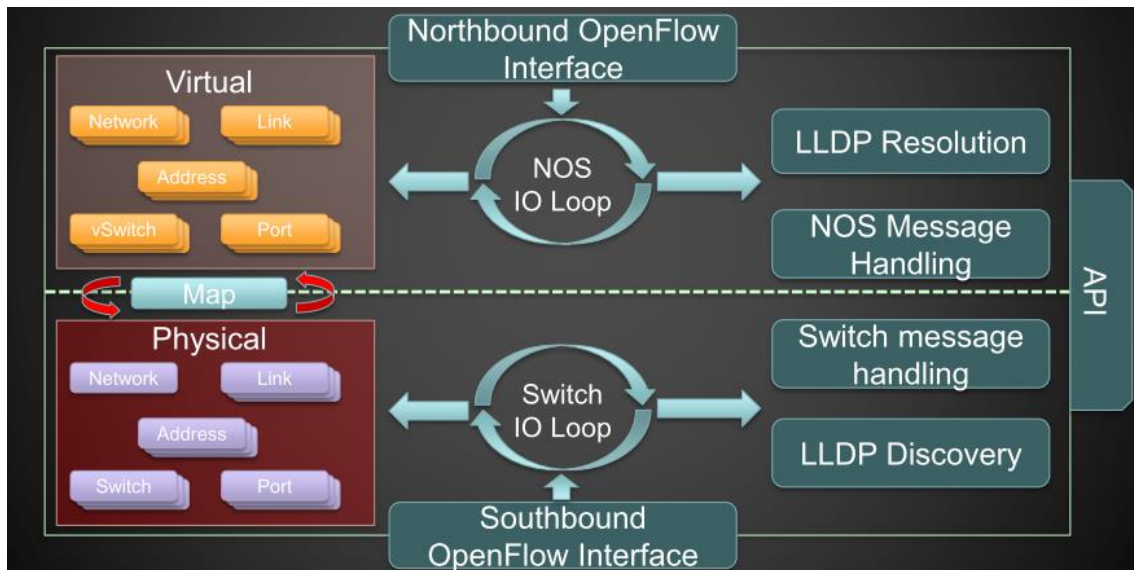


Fig. 14: OVN - Detalle componentes

Finalmente indicar sobre OpenVirtX que el proyecto a día de hoy parece, al igual que ocurre con Flowvisor, estar algo parado, lo cual da lugar a pensar que el proyecto vaya a dejar de estar soportado por sus creadores, los desarrolladores de ON.lab.

## Herramientas de creación de redes virtuales

De cara a la creación de los entornos de prueba virtuales se necesita alguna herramienta que permita la creación del entorno de red, de forma que sea flexible e implemente la mayor cantidad de funcionalidades posible. En este proyecto se han utilizado dos herramientas para cubrir esta necesidad: Mininet [29] y VNX [30]. Existen otras herramientas de emulación de redes virtuales, como NetKit, MarionNet, pero estos proyectos parecen estar bastante parados en los últimos años, por ello se utilizarán las dos herramientas indicadas anteriormente. Otras herramientas como Clownix, Core, GNS3 o IMUNES tampoco se utilizarán en este trabajo, pero se puede consultar más información sobre todas éstas en [31], un blog bastante interesante.

### Mininet

Mininet [29] es una herramienta de generación de escenarios de red virtuales muy utilizada en el ámbito de proyectos de investigación y desarrollo. De una forma bastante sencilla se pueden crear escenarios de red utilizando los comandos de Mininet o bien utilizando el API (en Python) de ésta. La última versión de Mininet es la 2.2.1 la cual fue liberada en Abril de 2015, y ya están preparando la siguiente versión, la 2.3. Este software es desarrollado principalmente por los desarrolladores Bob Lantz, Brandon Heller y Brian O'Connor, pero también han contribuido organizaciones como ON.lab (Stanford), Big Switch y Canonical.

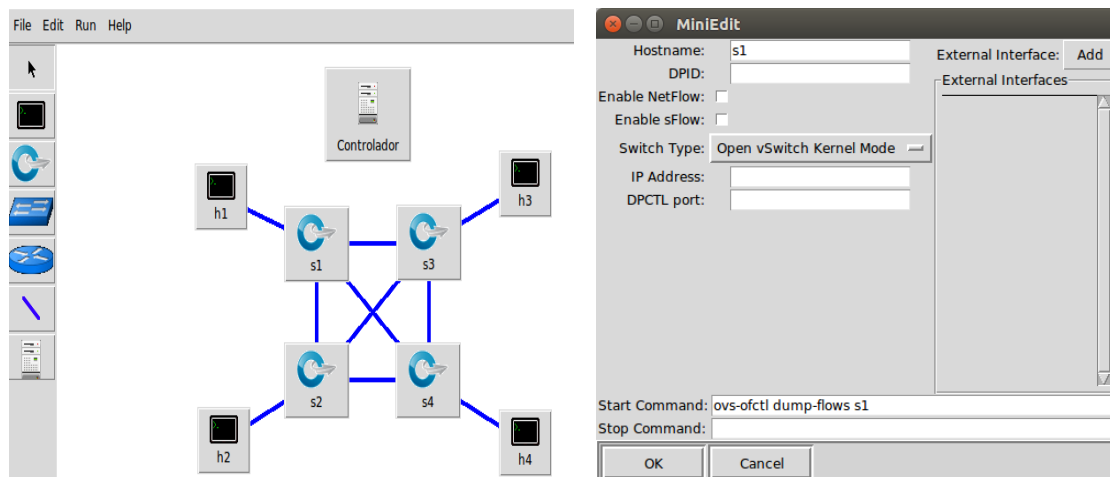
Esta herramienta se ejecuta sobre sistemas Linux, no obstante estos sistemas pueden ser virtuales, de forma que desde cualquier sistema operativo (OSx, Windows, etc.) podemos utilizar emuladores de máquinas virtuales para ejecutar una máquina virtual con Linux y poder ejecutar

Mininet. También existe la posibilidad de ejecutar Mininet en dispositivos *Raspberry Pi*, lo cual es interesante de cara a la implementación de escenarios basados en estos dispositivos.

Mininet utiliza los llamados *Linux network namespaces*[32] para la creación de nodos con espacios de red independientes (interfaces, tablas de rutas...), lo que le permite poder crear de forma rápida y eficiente decenas o cientos de nodos en un único equipo. Es decir, cada host o switch en Mininet, será un namespace diferente. El uso de recursos que utilizan estos nodos son los del propio equipo donde se ejecutan, pudiendo configurar la asignación en alguno de éstos (como la CPU por ejemplo). El resto de opciones que se pueden utilizar con estos *namespaces* estarán también disponibles en Mininet (configuración de interfaces, ejecución de comandos dentro de un *namespace* en concreto o en varios, conexión con interfaces físicas, etc.).

No dispone de interfaz gráfica de usuario *oficial*, por lo que la configuración de la herramienta se realiza normalmente desde la línea de comandos o mediante scripts en Python utilizando el API mencionado anteriormente. No obstante, existen algunos desarrollos de aplicaciones que nos ayudan, a través de interfaz gráfica, a generar los scripts de Mininet. Por ejemplo, existe una plataforma web, [33], en la que podemos dibujar la arquitectura de red y después generar el script correspondiente, que deberemos ejecutar posteriormente para crear el escenario de red.

Esta plataforma web no sólo está dedicada a Mininet, sino también a diferentes controladores SDN, incluido el proxy Flowvisor. Aunque algo limitada, esta herramienta web puede ser bastante útil sobre todo para los que están comenzando a investigar estas herramientas. Adicionalmente existe un desarrollo experimental que se incluye en los ejemplos de Mininet, se trata de *miniedit.py*, el cual ofrece una GUI para la creación de los escenarios de red bastante completo [34]. Desde esta interfaz se pueden configurar todos los parámetros de todos los dispositivos (hosts, switches y controladores), como su nombre, tipo de dispositivo, dirección IP, DPID, asignación de CPU...:



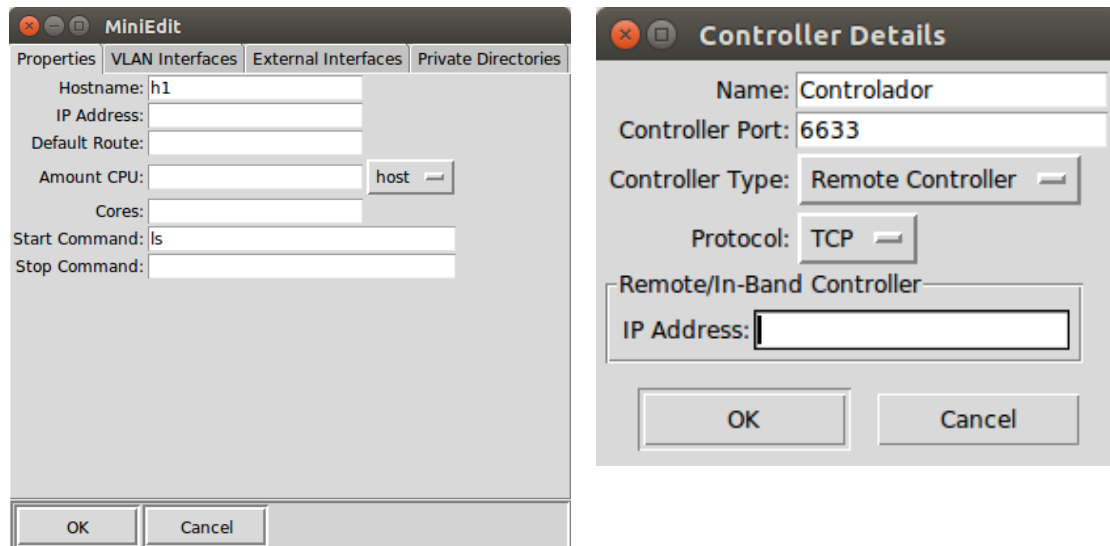


Fig. 15: miniedit - GUI experimental para Mininet

Los escenarios que se crean con *miniedit* se pueden exportar y guardarse para cargarlos posteriormente o también exportarlos en modo de script para directamente poder ejecutar el escenario.

## Instalación

En cuanto a la instalación, es bastante sencilla y desde la web oficial [29] ofrecen tres formas para comenzar a utilizar este software:

1. Mediante la descarga de una máquina virtual que ya tiene instalado todo lo necesario para ejecutar Mininet
  - Esta máquina virtual hay que ejecutarla bajo algún software de emulación como puede ser VirtualBox [35] o Qemu [36]
2. Instalación manual de Mininet utilizando las fuentes de la herramienta:
  - `git clone git://github.com/mininet/mininet`
  - `mininet/util/install.sh [options]`
  - Donde las principales opciones son:
    - `-a`: Instala Mininet y todo lo que está incluido en la máquina virtual de Mininet (Open vSwitch, Openflow Wireshark dissector y POX)
    - `-nfv`: Instala Mininet, switch de referencia OpenFlow y Open vSwitch
  - Con la opción `-h` se podrá ver más información sobre el resto de opciones.
  - Para realizar una primera prueba y comprobar que todo funciona bien, bastaría con ejecutar:
    - `sudo mn --test pingall`
3. Instalación manual utilizando paquetes:
  - `sudo apt-get install Mininet`
  - Habría ahora que deshabilitar la ejecución del controlador que se instala por defecto para openVswitch:
    - `sudo service openvswitch-controller stop`

- `sudo update-rc.d openvswitch-controller disable`
- Para realizar una primera prueba y comprobar que todo funciona bien, bastaría con ejecutar:
  - `sudo mn --test pingall`

Más adelante en la descripción de los escenarios de red propuestos, se podrán ver las diferentes configuraciones realizadas con Mininet, que en este caso serán implementadas con las APIs de Python que ofrece la herramienta.

## VNX: Virtual Networks over Linux

Esta herramienta [30] ha sido desarrollada por el Departamento de Ingeniería Telemática (DIT) en la Escuela Técnica Superior de Ingenieros de Telecomunicación (ETSIT) de la Universidad Politécnica de Madrid (UPM). En base a ficheros de configuración XML permite la creación y despliegue automático de escenarios virtuales de red de una forma sencilla. Con el uso de contenedores Linux (LXC) se pueden crear diferentes hosts (Linux, Windows, etc.) conectados entre sí tal y como el usuario lo haya configurado. Para la emulación de los diferentes hosts, es necesario descargar los sistemas de ficheros correspondientes (rootfs), y en base a esos rootfs, el contenedor LXC será de una naturaleza u otra.

VNX también es capaz de emular de forma limitada las funcionalidades de routers Cisco o Juniper, gracias a la integración con las plataformas de virtualización de Dynamips y Olive, lo cual extiende aún más las funcionalidades de VNX.

Otro punto a destacar de VNX es que está en constante evolución introduciendo mejoras y nuevas funcionalidades, por lo que al ser un proyecto vivo le da ventajas frente a otros que al parecer llevan parados algunos años (NetKit o MarionNet, por ejemplo).

La instalación de esta herramienta, si se hace paso a paso, es algo más tediosa que en el caso de Mininet, pero no por ello es complicada. No obstante, también se ofrece la posibilidad de utilizar una máquina virtual aprovisionada con Vagrant donde ya está todo instalado y preparado para poder ejecutar entornos de red con VNX.

Esta herramienta permite algunas otras opciones interesantes, como es la creación de espacios de nombres para ejecución de secuencias de comandos, de forma que una vez definidos y con el entorno operativo, se pueden enviar esas secuencias de comandos que se ejecutarán en los hosts donde se hayan definido.

Otra característica a destacar es la posibilidad de integrar el propio host donde se ejecuta en entorno como parte del entorno, lo que posibilita, por ejemplo, el uso de navegadores web desde el host (en los contenedores LXC normalmente no hay entorno gráfico).

Entre las herramientas que dispone cabe destacar también la que permite generar un mapa de la red que se ha configurado, de manera que de forma visual se puede ver el escenario creado lo que permite por ejemplo corroborar fácilmente todas las conexiones entre las diferentes redes y los hosts asociados a éstas:

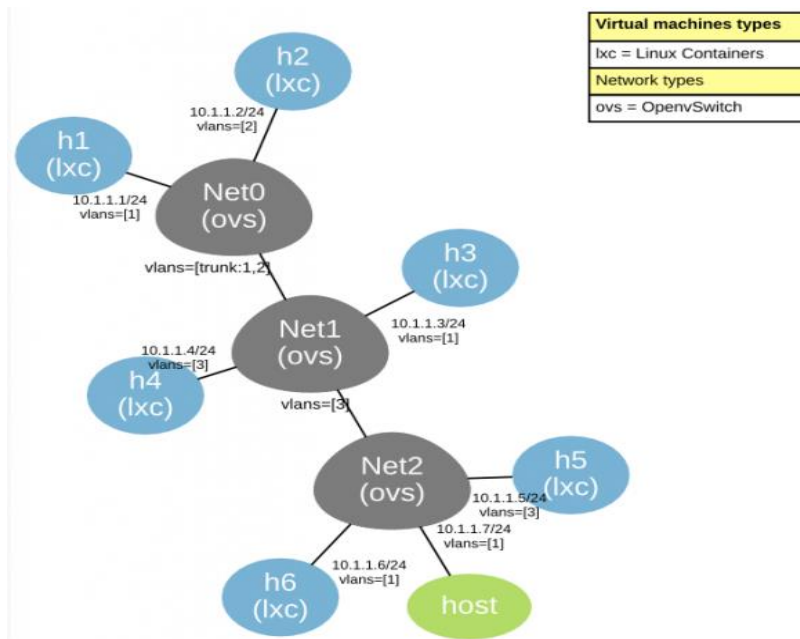


Fig. 16: VNX - Mapa escenario de red (ejemplo con VLANs)

## Instalación

Para la instalación de esta herramienta se recomienda el uso del sistema operativo Ubuntu server versión 10.04 o superior y que el prosador tenga soporte de virtualización (hoy día prácticamente todos soportan esta característica).

La instalación mediante paquetes aún no está disponible, de manera que hay que seguir una serie de pasos e instalar y descargar diferente software para tener el entorno VNX operativo. En primer lugar, hay que instalar (si no los tenemos ya instalados) una serie de paquetes, que podemos instalar con el siguiente comando:

- `sudo apt-get install qemu-kvm libvirt-bin vlan xterm bridge-utils screen virt-manager`

Una vez instalado hay que realizar algunos cambios en la configuración de *libvirt* para que funcione con VNX. Para ello hay que modificar el fichero `/etc/libvirt/qemu.conf` añadiendo `/dev/net/tun` a este fichero y reiniciar *libvirtd*:

- ```
security_driver = "none"
user = "root"
group = "root"
cgroup_device_acl = [
    "/dev/null", "/dev/full", "/dev/zero",
    "/dev/random", "/dev/urandom",
    "/dev/ptmx", "/dev/kvm", "/dev/kqemu",
    "/dev/rtc", "/dev/hpet", "/dev/vfio/vfio", "/dev/net/tun"
]
```
- `sudo restart libvirt-bin` # for ubuntu 14.10 or older
- `sudo systemctl restart libvirt-bin` # for ubuntu 15.04 or later

A continuación, con el entorno ya preparado, se procede a instalar el software de VNX propiamente dicho. Para ello, basta con seguir estos pasos:

- `mkdir /tmp/vnx-update`
- `cd /tmp/vnx-update`
- `rm -rf /tmp/vnx-update/vnx-*`
- `wget http://vnx.dit.upm.es/vnx/vnx-latest.tgz`
- `tar xfvz vnx-latest.tgz`
- `cd vnx-*`
- `sudo ./install_vnx`
- Crear un fichero nuevo de configuración, se puede utilizar uno de los ejemplos:
 - `sudo mv /usr/share/vnx/etc/vnx.conf.sample /etc/vnx.conf`

Una vez instalado, hay que descargar el *rootfs* que utilizarán los contenedores LXC de los que hace uso VNX. Estas intrucciones se pueden encontrar en la url:

- [http://web.dit.upm.es/vnxwiki/index.php/Vnx-install-root fs](http://web.dit.upm.es/vnxwiki/index.php/Vnx-install-root_fs)

Adicionalmente se puede realizar la instalación para poder crear escenarios compatibles con *Dynamips*. En este trabajo no se va a utilizar, por lo que no se detallan los pasos para hacerlo, no obstante en la documentación del proyecto VNX se detallan todos los pasos.

Comando útiles de VNX

A continuación se indican algunos comandos útiles que se utilizarán con toda seguridad en cualquier despliegue de VNX:

- `sudo vnx -f escenario.xml -v -t`
 - Inicia el escenario
- `sudo vnx -f escenario.xml --console-info`
 - Muestra la información de las consolas del escenario desplegado
- `sudo vnx -f escenario.xml -v --exe calc`
 - Inicia la calculadora en la máquina virtual donde se haya configurado
- `sudo vnx -f escenario.xml -v --exe calcoff`
 - Cierra la calculadora (*kill*) en la máquina virtual
- `sudo vnx -f escenario .xml -v --show-map`
 - Muestra un mapa con la arquitectura de red configurada en el escenario
- `ping -c 4 10.0.0.1`
 - Comando *ping* para comprobación de la conectividad de la VM
- `sudo vnx -f escenario.xml -v --shutdown`
 - Cierra el escenario guardando los cambios realizados en el mismo
- `sudo vnx -f escenario.xml -v --destroy`
 - Cierra el escenario sin guardar los cambios en el mismo

Otra característica a destacar del proyecto VNX es la gran cantidad de ejemplos ya implementados que ayudan bastante a la hora de conocer y poner en funcionamiento escenarios personalizados. Una lista de estos ejemplos está disponible en la sección de *Examples* de la web del proyecto.

CAPÍTULO 3: ESCENARIOS

De cara a crear los escenarios donde se puedan estudiar los conceptos de las arquitecturas SDN, existen varias herramientas que son capaces de crear todo tipo de dispositivos y topologías de red. Así, existen soluciones como Mininet, muy utilizada en proyectos de investigación y de pruebas, y con la que se pueden crear hosts, switches, routers e interconectarlos para crear una red con la que realizar pruebas. Otra buena herramienta para generar escenarios es VNX, la cual basándose en ficheros de configuración XML, crea contenedores virtuales Linux (tecnología LXC [37]) que actúan como hosts o elementos de red.

Herramientas utilizadas

Para la implementación de los escenarios se han utilizado las herramientas de Mininet y VNX de cara a la creación de los entornos de red. Como controlador de SDN se utilizará Floodlight y como proxy de estos controladores y para conseguir delegar algunas partes de la red y/o del tráfico que por ésta pase, se utilizará Flowvisor.

De cara a poder facilitar y optimizar el uso de las herramientas indicadas anteriormente para crear los escenarios que veremos a continuación, se ha hecho uso de los llamados *contenedores Linux* (LXC) para la simulación de equipos, switches virtuales y controladores SDN. Para la comunicación entre todos los LXC, se utilizan los *bridges* [38] de Linux, de forma que se puede simular cualquier tipo de escenario de red intercomunicado como convenga.

En el caso de la implementación de escenarios con Mininet, el uso de LXC ha sido implementado como parte del trabajo para mejorar la creación de escenarios y evitar así problemas como puede ser el uso de dos instancias del mismo controlador SDN sobre el mismo equipo, ya que hay que cambiar la configuración de éstos para que no utilicen los mismos puertos, o el mismo directorio para la base de datos.

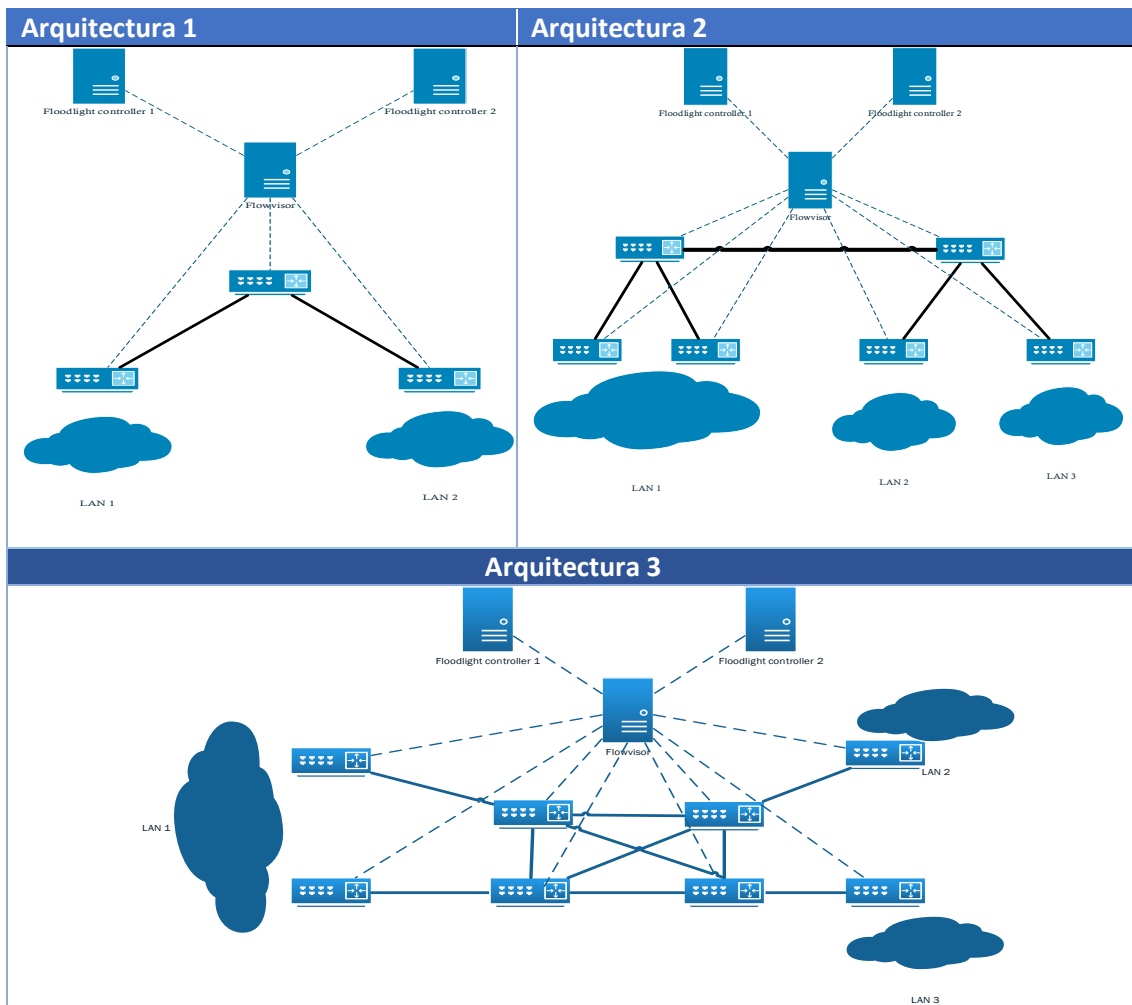
En cambio, en los mismos escenarios implementados con VNX no ha sido necesario implementar nada adicional para utilizar estos contenedores, ya que la propia herramienta ya hace uso de ellos.

Escenarios propuestos

Se han configurado algunos escenarios de red con diferente nivel de complejidad de cara a que puedan ser utilizados con fines académicos y que los alumnos conozcan las características de las arquitecturas SDN y la integración de Flowvisor en éstas. Todos los escenarios utilizan las mismas herramientas: Floodlight y Flowvisor. Todos los escenarios se han creado tanto con Mininet como con VNX.

Las arquitecturas de red de los escenarios propuestos se pueden dividir en tres arquitecturas base, sobre las cuales se han definido una serie de condiciones que harán que la configuración del Flowvisor y Floodlight tenga mayor o menor complejidad.

Las tres arquitecturas base son las siguientes:



Según las condiciones que se establezcan, serán necesario realizar la configuración de unas reglas u otras. Por ejemplo, cualquiera de las arquitecturas además de configurar las reglas correspondientes para que los equipos en LAN1 conecten con los equipos en LAN2 o LAN3, se podrían establecer condiciones de uso de QoS en función de ciertas características del tráfico (puerto TCP destino, subred origen/destino, tipo de protocolo de red, etc).

Escenario 1

Este escenario simple se propone como primera toma de contacto para los alumnos con las arquitecturas SDN y con Flowvisor. Para este escenario se utilizará como base la arquitectura 1, y se propone configurar Flowvisor y Floodlight (solo se utilizará un controlador en este escenario) para que haya conexión entre los equipos conectados al switch S2 y los conectados al switch S3. De cara a conocer un poco mejor el módulo de *Forwarding* que ofrece Floodlight, se plantean dos configuraciones sobre el mismo escenario: Una iniciando el controlador Floodlight con el módulo de *forwarding* activado y otra sin él. El esquema de este escenario quedaría como sigue:

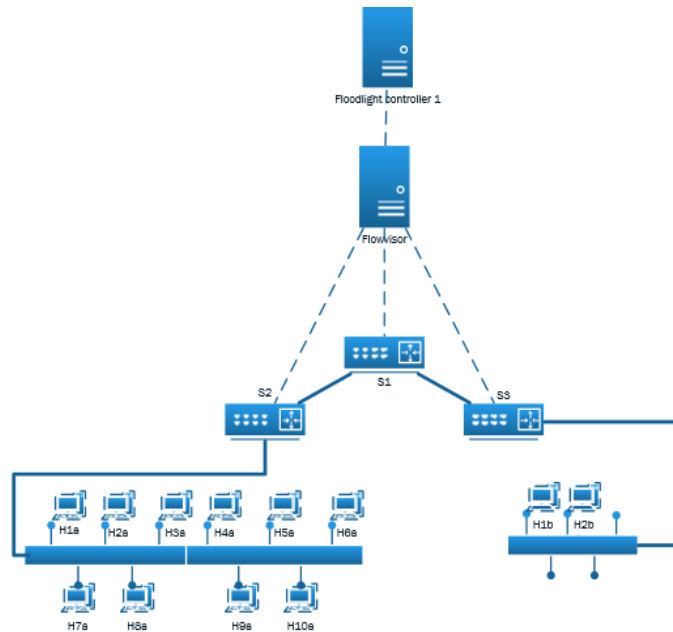


Fig. 17: Escenario 1

De esta forma, y como se verá en el apartado de configuraciones, en el escenario en el que está el módulo de forwarding activado, se podrá ver cómo se van configurando los flujos automáticamente en todos los switches a medida que generamos tráfico entre los hosts virtuales (siempre que previamente se haya configurado la slice y los flowspaces correspondientes en Flowvisor). En cambio, sin este módulo activado, no habrá conectividad entre éstos aun cuando se estén ya configurados los flowspaces y la slice en Flowvisor, a menos que se configuren también las reglas correspondientes en el controlador de Floodlight.

En este escenario, la distribución de direcciones, identificadores y puertos de nivel de transporte utilizados se detalla en la siguiente tabla:

Equipo	Dirección IP/Mask	DPID	Puertos transporte
H1a	10.0.0.1/23	N/A	N/A
H2a	10.0.0.2/23	N/A	N/A
H3a	10.0.0.3/23	N/A	N/A
H4a	10.0.0.4/23	N/A	N/A
H5a	10.0.0.5/23	N/A	N/A
H6a	10.0.0.6/23	N/A	N/A
H7a	10.0.0.7/23	N/A	N/A
H8a	10.0.0.8/23	N/A	N/A
H9a	10.0.0.9/23	N/A	N/A
H10a	10.0.0.10/23	N/A	N/A
H1b	10.0.2.1/24	N/A	N/A
H2b	10.0.3.1/24	N/A	N/A
S1	N/A	00:00:00:00:00:00:01	N/A
S2	N/A	00:00:00:00:00:00:02	N/A
S3	N/A	00:00:00:00:00:00:03	N/A
Flowvisor	10.15.0.2	N/A	Openflow: 6633 Admin: 8081
Floodlight 1	10.15.0.3	N/A	Openflow: 6653 Web UI: 8082

Tabla 2: Direccionamiento Escenario 1

A continuación se indica un posible guion que puede servir de base para un enunciado donde se utilizaría el escenario anterior. Se supone que al alumno se le proporciona un entorno (máquina virtual, liveCD, etc.) en el que los escenarios de red ya están creados (con VNX, por ejemplo) y se le dará las instrucciones convenientes para que pueda ejecutarlo.

- La empresa Ea que cuenta con diferentes sedes repartidas por la ciudad, necesita que los equipos de su red principal (LAN 1) puedan tener conexión a dos servidores que tiene en otras dos sedes diferentes. Para ello, esta empresa quiere implementar una infraestructura SDN como la indicada más arriba (imagen y tabla anteriores), donde se utilizarán 3 switches con capacidad de Openflow (open vswitch), un proxy de Openflow (Flowvisor) y un controlador de Openflow, Floodlight. Para ello, y utilizando el entorno dado por el DIT y las instrucciones de uso adjuntas, se pide:
 - - i. Levantar el escenario 1 utilizando el controlador Floodlight **con** el módulo de *forwarding* activado.
 - ii. Configurar la slice para este controlador Floodlight en Flowvisor así como los flowspaces necesarios. A continuación se muestra un ejemplo de creación de una slice y un flowspace asociado a ésta en Flowvisor:
 - Creación slice:
 - `fvctl -f /etc/flowvisor.passwd -h IP_Flowvisor -p 8081 add-slice slice_name -p fvadmin tcp:IP_CONTROLLER:PORT_CONTROLLER sliceCa@tfm.es`
 - Creación flowspace (con permiso escritura para slice1):
 - `fvctl -f /etc/flowvisor.passwd -h IP_FLOWVISOR -p 8081 add-flowspace FS_NAME any 110 any slice1=4`
 - iii. Comprobar la conectividad entre los diferentes equipos y ver las reglas de flujo de cada switch (ver comando `ovs-ofctl`)
 - iv. Levantar el escenario 1 utilizando el controlador Floodlight **sin** el módulo de *forwarding* activado.
 - v. Con la configuración de slice y flowspaces del apartado anterior, añadir la configuración de las reglas de flujo en Floodlight necesarias para conseguir conectividad entre todos los equipos.
 - vi. Comprobar la conectividad entre los diferentes equipos, ver las reglas de flujo de cada switch (ver comando `ovs-ofctl`) y compararlas con las del apartado A.

Con este escenario y con este enunciado de ejemplo, una vez que el alumno lo haya resuelto conocerá las bases de una arquitectura SDN, así como el funcionamiento básico de las herramientas de Floodlight y Flowvisor. En los siguientes escenarios, se aumenta la complejidad de configuración tanto de Flowvisor como de Floodlight, introduciendo el uso de más funcionalidades de Flowvisor y la necesidad de configurar más reglas en Floodlight.

Escenario 2

Este escenario utilizará la arquitectura 2 y se van a configurar tres grupos de equipos que pertenecerán a tres subredes diferentes. La idea de este escenario es conseguir dividir la red en dos espacios diferentes diferenciados entre sí por la red destino a la que se quiere acceder y **delegar el control de esos dos espacios de red en diferentes controladores SDN**. Además, en los switches centrales (S1 y S6), se compartirán los puertos de entrada y salida de tráfico entre

las dos particiones de red, pero debido al resto de condiciones de los flowspaces, cada controlador sólo gestionará los flujos que les correspondan. Esta división se llevará a cabo con Flowvisor, con la creación de dos slices y los flowspaces necesarios, por lo tanto, utilizaremos en este caso dos controladores Floodlight (uno por slice). En este ejemplo, el módulo de *Forwarding* de los controladores Floodlight estará deshabilitado. Además, el alumno deberá revisar en el texto del estándar de Openflow las dependencias de los parámetros de filtrado que va a utilizar, ya que los parámetros para filtrar los flujos en base a red origen o destino deben definirse junto con el parámetro *dl_type* correspondiente (para IPv4, o para ICMP, por ejemplo, tendrá que establecer los valores correspondientes). Por otro lado, en esta propuesta también se introduce al alumno a interactuar con la administración de los switchs Openflow a través de la línea de comandos no sólo para consultar datos (como las tablas de flujos), sino también para realizar la configuración de las colas QoS.

En condiciones similares a las del anterior escenario, un posible guion para un enunciado que utiliza este escenario sería el siguiente:

- El proveedor de servicios de red SDNISP se dispone a dar servicios de conexiones entre diferentes sedes, similar a lo que hoy hacemos con una VPN, pero con las facilidades y escalabilidad de una arquitectura SDN con Flowvisor. Para ello utiliza dispositivos de red con capacidades de Openflow que se configurarán de forma que el tráfico de los diferentes circuitos contratados por los clientes viajará a través del mismo (o mismos) enlace físicos pero sin interferencia entre ellos. Además, este ISP ofrece diferentes opciones de conexión en cuanto al ancho de banda contratado, por lo que cada cliente conectará con sus otras sedes utilizando el ancho de banda contratado. La arquitectura de red que dispone este ISP para los escenarios que se plantean después, es la siguiente:

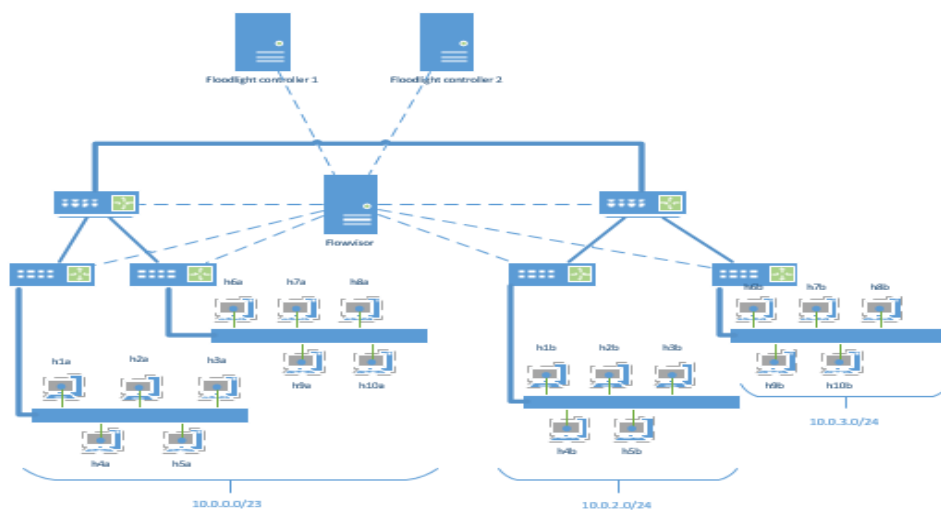


Fig. 18: Escenario 27

Tabla 3: Direccionamiento e identificadores Escenario 2

Equipo	Dirección IP/Mask	DPID	Puertos transporte
H1a	10.0.0.1/23	N/A	N/A
H2a	10.0.0.2/23	N/A	N/A
H3a	10.0.0.3/23	N/A	N/A

H4a	10.0.0.4/23	N/A	N/A
H5a	10.0.0.5/23	N/A	N/A
H6a	10.0.0.6/23	N/A	N/A
H7a	10.0.0.7/23	N/A	N/A
H8a	10.0.0.8/23	N/A	N/A
H9a	10.0.0.9/23	N/A	N/A
H10a	10.0.0.10/23	N/A	N/A
H1b	10.0.2.1/24	N/A	N/A
H2b	10.0.2.2/24	N/A	N/A
H3b	10.0.2.3/24	N/A	N/A
H4b	10.0.2.4/24	N/A	N/A
H5b	10.0.2.5/24	N/A	N/A
H6b	10.0.3.1/24	N/A	N/A
H7b	10.0.3.2/24	N/A	N/A
H8b	10.0.3.3/24	N/A	N/A
H9b	10.0.3.4/24	N/A	N/A
H10b	10.0.3.5/24	N/A	N/A
S1	N/A	00:00:00:00:00:00:00:01	N/A
S2	N/A	00:00:00:00:00:00:00:02	N/A
S3	N/A	00:00:00:00:00:00:00:03	N/A
S4	N/A	00:00:00:00:00:00:00:04	N/A
S5	N/A	00:00:00:00:00:00:00:05	N/A
S6	N/A	00:00:00:00:00:00:00:06	N/A
Flowvisor	10.15.0.2	N/A	Openflow: 6633 Admin: 8081
Floodlight 1	10.15.0.3	N/A	Openflow: 6653 Web UI: 8082
Floodlight 2	10.15.0.4	N/A	Openflow: 6653 Web UI: 8082

Cada circuito diferente estará asociado y por tanto será gestionado por un controlador de Openflow diferente.

- En el primer escenario, se dispone de un cliente, ClienteA (Ca) el cual dispone de una sede en Madrid y quiere conectar con las sedes de dos de sus clientes situadas en Valencia y Salamanca respectivamente. El Ca quiere que su circuito tenga un ancho de banda de 10Mbps (cola q0) para conectar con Valencia y de 20Mbps (cola q1) para conectar con el cliente de Salamanca. En la sede de Madrid, el Ca tiene un direccionamiento de ipv4 10.0.0.0/23 (endpoints hXa), en Valencia 10.0.2.0/24 (endpoints h1b a h5b) y en Salamanca 10.0.3.0/24 (endpoints h6b a h10b). Utilizando la VM proporcionada por el DIT, y haciendo uso de los anexos de la práctica, se pide:
 - i. Iniciar el escenario de red con Mininet/VNX utilizando la ayuda del Anexo 1.
 - ii. Configurar las colas de QoS en los puertos de los switches S1 y S6 (ayuda en el Anexo 2). Aportar la configuración de las colas de ambos switches (ver comando ovs-vsctl)
 - iii. Ejecución del proxy de los controladores (Flowvisor) y de los dos

controladores. Encontrará ayuda para el arranque de estos componentes en el Anexo 3.

- iv. Configuración, mediante scripts, de las Slices y Flowspaces en Flowvisor para cada controlador de forma que cada controlador sólo tenga acceso al tráfico de su VPN independientemente de lo que se pueda configurar en los controladores. Una vez configurado, se podrá ver tanto en los logs de los controladores como en la GUI los switches a los que tienen acceso (aportar pantallazos de esta parte así como los ficheros de configuración generados)
- v. Configuración, mediante scripts y usando el api REST de los controladores Floodlight, de las reglas de flujo que cada controlador va a establecer en los flowspaces configurados anteriormente.
- vi. Una vez ejecutado el/los scripts, comprobar la conectividad de las dos VPNs y aportar tanto las tablas de flujo de cada switch (ver comando `ovs-ofctl`) como los pings entre endpoints de la misma VPN y endpoints de diferentes VPNs.

Una vez resuelto el escenario anterior, el alumno tendrá conocimientos más avanzados sobre cómo opera una arquitectura SDN así como de la lógica de funcionamiento de un controlador como Floodlight y del proxy Openflow Flowvisor. Con lo planteado anteriormente, se confirman también las ventajas que ofrecen estas herramientas, de cara a disponer de una administración centralizada (en los controladores) sobre el tráfico que Flowvisor presenta en cada slice. Además, de cara a realizar esta práctica en grupo (en parejas por ejemplo), una vez que Flowvisor esté configurado, cada alumno se encargaría de uno de los controladores, de forma que haga lo que haga no afectará a la configuración ni a los flujos correspondientes al controlador que configuraría el otro alumno.

En el siguiente ejemplo, se propone sobre la misma arquitectura una práctica diferente con un caso de uso diferente, y tendrá algunos detalles en los que el alumno tendrá que volver revisar el estándar de Openflow para resolverlos, no obstante es un paso más en el aprendizaje y la comprensión de las capacidades de Flowvisor, Openflow y las arquitecturas SDN.

Escenario 3

En este último escenario la arquitectura de red será la arquitectura 3 mostrada anteriormente. La idea de este tercer escenario es utilizar de nuevo las capacidades de particionado de la red con Flowvisor. En esta ocasión la finalidad del escenario se centrará en la necesidad del uso la virtualización para poder delegar de forma segura la administración de diferentes redes, de forma que estas sean configurables de forma dinámica y en tiempo real según las necesidades de su entorno, en concreto, según las necesidades de las aplicaciones desde las cuales se van a configurar. Con esta idea, se creará con Flowvisor de nuevo dos slices que presentarán diferentes switches a cada controlador Floodlight (de nuevo, dos controladores, uno por slice), de forma que habrá diferentes caminos disponibles para llegar al mismo destino. Se van a mantener en estos flowspaces las reglas de filtrado por red origen y destino, lo cual, en los switches comunes a ambos, ofrecerá la seguridad de que las aplicaciones que configuran la red desde un controlador no afectarán a las configuraciones hechas desde el otro. A continuación

se presenta de nuevo un guion que puede servir de referencia de cara a plantear un enunciado que engloba a este escenario y a su finalidad.

- Una empresa de ingeniería necesita implementar diferentes redes para dar servicio a los trabajos de dos de sus departamentos. Estos departamentos se dedican al desarrollo e implementación de nuevos protocolos de comunicación para las soluciones hardware/software propietarios que tiene. Estos desarrollos necesitan tener visibilidad de la red por la que envían los datos, de forma que puedan variar su configuración de forma dinámica en base a diferentes parámetros (creación de colas QoS en los diferentes puertos y envío del tráfico por una u otra según convenga por ejemplo). Para la implementación de este escenario, se dispone de la siguiente arquitectura de red:

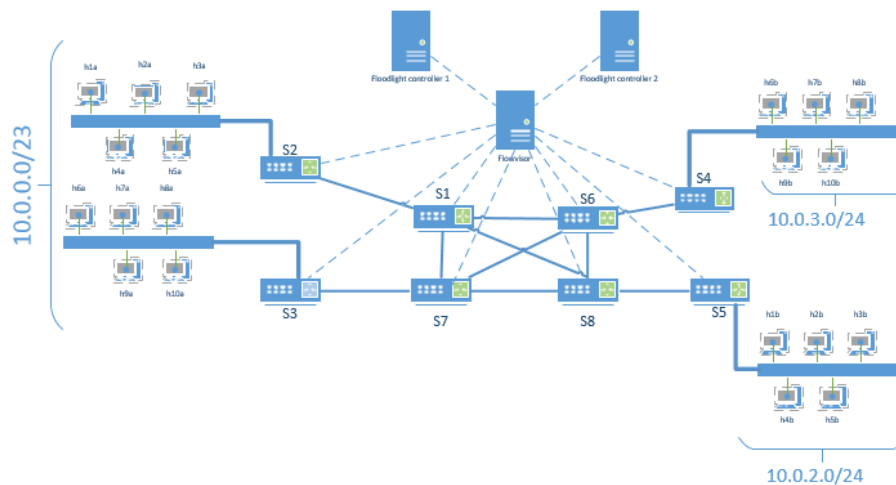


Fig. 19: Escenario 3

El direccionamiento es igual que en el escenario 2, añadiendo la siguiente información sobre los nuevos switches de la arquitectura:

Equipo	Dirección IP/Mask	DPID	Puertos transporte
S7	N/A	00:00:00:00:00:00:07	N/A
S8	N/A	00:00:00:00:00:00:08	N/A

Las dos slices que se van a crear tienen las siguientes condiciones:

- RED 1
 - Switchs S2,S1,S6,S8 y S4
 - Red origen: 10.0.0.0/23
 - Red destino: 10.0.2.0/24
- RED 2
 - Switch S3,S5,S6,S7 y S8
 - Red origen: 10.0.0.0/23
 - Red destino: 10.0.3.0/24

En primer lugar, el tráfico de la RED 1, deberá seguir el camino:

- S1 <-> S1 <-> S6 <-> S4

Y en la RED 2:

- S3 <-> S7 <-> S8 <-> S5

- Utilizando la VM proporcionada por el DIT, y haciendo uso de los anexos de la práctica, se pide:
 - - i. Iniciar el escenario de red con Mininet/VNX utilizando la ayuda del Anexo 1
 - ii. Ejecución del proxy de los controladores (Flowvisor) y de los dos controladores (uno para cada VPN). Encontrará ayuda para el arranque de estos componentes en el Anexo 3.
 - iii. Configuración, mediante scripts, de las Slices y Flowspaces en Flowvisor para cada controlador de forma que cada controlador sólo tenga acceso al tráfico de su VPN independientemente de lo que se pueda configurar en los controladores. Una vez configurado, se podrá ver tanto en los logs de los controladores como en la GUI los switches a los que tienen acceso. Aportar pantallazos de esta parte así como los ficheros de configuración generados.
 - iv. Configuración, mediante scripts y usando el api REST de los controladores Floodlight, de las reglas de flujo que cada controlador va a establecer en los flowspaces configurados anteriormente. Una vez ejecutado el/los scripts, comprobar la conectividad de los equipos y aportar tanto las tablas de flujo de cada switch (ver comando `ovs-ofctl`)
 - v. Codificar una aplicación web (libre elección del lenguaje de programación) que ofrezca una GUI con posibilidades para:
 - Configurar reglas en los controladores mediante elección de parámetros mediante formularios
 - Las reglas podrán ser tanto añadidas como eliminadas
 - Cambiar el camino que sigue el tráfico de cada slice definida de forma que ahora sea:
 - RED 1: S1 <-> S1 <-> S8 <-> S6 <-> S4
 - RED 2: S3 <-> S7 <-> <-> S6 <-> S8 <-> S5
- Comprobar si durante el cambio se producen pérdidas de paquetes (ping continuo mientras se hacen los cambios)

Una vez finalizado el escenario anterior (se supone que los dos primeros también), el alumno tendrá conocimientos bien asentados sobre las arquitecturas SDN y las características adicionales que la herramienta Flowvisor ofrece. Además, habrá conocido más de cerca la configuración de Openflow y de algunos comandos útiles de cara a la consulta y configuración de los *open vswitches*.

Configuración y pruebas

A continuación se exponen las soluciones implementadas a los escenarios propuestos anteriormente (excepto la aplicación web del tercer escenario, que queda fuera del alcance de este trabajo, pero se presenta el código necesario para realizar los cambios de la configuración)

El código de la configuración implementada en Flowvisor y Floodlight, así como la configuración de los escenarios de red desplegados con Mininet y VNX se puede ver en el [Anexo I](#), [Anexo II](#) y [Anexo III](#) para los escenarios 1, 2 y 3 respectivamente. A continuación se exponen algunas capturas de pantalla con los resultados obtenidos en los diferentes escenarios.

En todos los escenarios se han creado scripts de arranque y parada de los mismos, los cuales son los ficheros llamados *start*.sh* y *stop*.sh*. En estos scripts se lanzan los diferentes escenarios y además, al haber utilizado dos herramientas que crean bridges con diferente nombre, se han añadido también líneas que añaden o eliminan rutas en el host según convenga.

Junto con la memoria, además de todos los ficheros cuyo contenido se expone a continuación, también se entregan los contenedores LXC utilizados, de cara a que puedan ser utilizados en otro equipo en un futuro si se desea.

Escenario 1

El gráfico de red que se obtiene con la herramienta VNX es el siguiente:

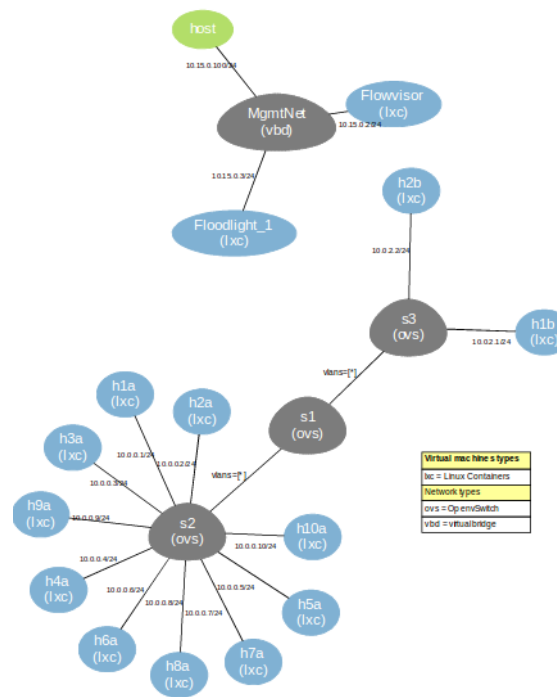


Fig. 20: Mapa red escenario 1 con VNX

Donde se distinguen los distintos switches (todos son open vswitch) así como los sistemas conectados a cada uno de éstos. Se ha creado también una red independiente de gestión que es donde está conectados el controlador SDN, Flowvisor y la propia máquina desde donde se está ejecutando (esto es útil para por ejemplo poder acceder vía web al controlador Floodlight).

En la siguiente imagen se ve cómo se auto configuran los flowspaces (enmarcado en verde) cuando se hace un ping entre diferentes hosts (en la captura se estaba haciendo un ping entre h1a y h1b), una vez deja de haber tráfico (segundo marco rojo), se eliminan las reglas de las tablas de flujos. En la captura se monitoriza la tabla de flujos del switch S1:

```

root@javiubuntu:/home/administrador/tfm_jcm/escenarios/vpn/demo0/lxc# ovs-ofctl
dump-flows s1
NXST_FLOW reply (xid=0x4):
root@javiubuntu:/home/administrador/tfm_jcm/escenarios/vpn/demo0/lxc# ovs-ofctl
dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x101, duration=2.658s, table=0, n_packets=1, n_bytes=42, idle_timeout=5
, idle_age=2, priority=1,arp,in_port=2,dl_src=00:00:00:00:00:1b,dl_dst=00:00:00:
00:00:1a actions=output:1
 cookie=0x102, duration=2.651s, table=0, n_packets=3, n_bytes=294, idle_timeout=
5, idle_age=0, priority=1,ip,in_port=1,dl_src=00:00:00:00:00:1a,dl_dst=00:00:00:
00:00:1b,nw_src=10.0.0.1,nw_dst=10.0.2.1 actions=output:2
 cookie=0x103, duration=2.641s, table=0, n_packets=3, n_bytes=294, idle_timeout=
5, idle_age=0, priority=1,ip,in_port=2,dl_src=00:00:00:00:00:1b,dl_dst=00:00:00:
00:00:1a,nw_src=10.0.2.1,nw_dst=10.0.0.1 actions=output:1
root@javiubuntu:/home/administrador/tfm_jcm/escenarios/vpn/demo0/lxc# ovs-ofctl
dump-flows s1
NXST_FLOW reply (xid=0x4):
root@javiubuntu:/home/administrador/tfm_jcm/escenarios/vpn/demo0/lxc#

```

Este mismo escenario se implementa también sin el módulo de forwarding activado, de forma que es necesario configurar las reglas en el controlador Floodlight que permitan la comunicación entre los diferentes equipos. Una vez establecidas, las tablas de flujos de los tres switches quedan como sigue:

Tabla de flujos de S1:

```

NXST_FLOW reply (xid=0x4):
 cookie=0x101, duration=84.114s, table=0, n_packets=285, n_bytes=37899, idl

```

Tabla de flujos de S2:

```

NXST_FLOW reply (xid=0x4):
 cookie=0x101, duration=138.664s, table=0, n_packets=367, n_bytes=51877, idl
output:5,output:6,output:7,output:8,output:9,output:10,output:11,LOCAL

```

Tabla de flujos de S3:

```

NXST_FLOW reply (xid=0x4):
 cookie=0x101, duration=191.617s, table=0, n_packets=375, n_bytes=51882, idl

```

Escenario 2

El mapa generado con la herramienta VNX en este caso quedaría como sigue:

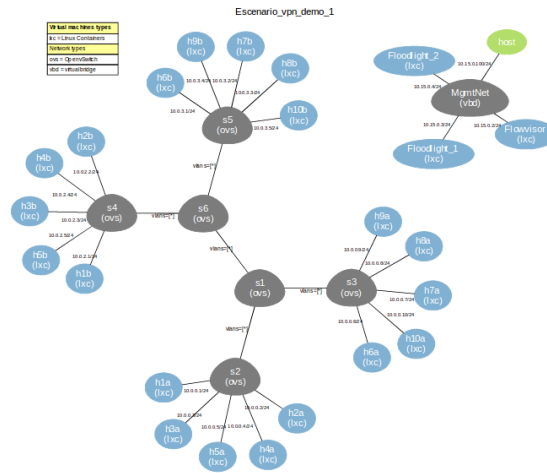


Fig. 21: Mapa red Escenario 2 con VNX

En este escenario intervienen no sólo más elementos de red sino más reglas que hacen que la configuración ya no sea tan sencilla como en el escenario anterior. En las imágenes siguientes se puede ver el resultado de haber asignado a dos tipos de tráfico diferente (diferenciados por red de destino) una cola QoS diferente. La ventaja de usar aquí la virtualización es de nuevo la centralización de la configuración y el aislamiento entre las reglas que afectan a un tipo de tráfico y a otro:

```

root@h1a:/home/ubuntu# iperf -c h4b -p 85
-----
Client connecting to h4b, TCP port 85
TCP window size: 85.3 KByte (default)
-----
[ 3] local 10.0.0.1 port 48505 connected with 10.0.2.4 port 85
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-10.6 sec  14.0 MBytes 11.1 Mbits/sec
root@h1a:/home/ubuntu#

```

```

root@h4b:/home/ubuntu# iperf -s -p 85
-----
Server listening on TCP port 85
TCP window size: 85.3 KByte (default)
-----
[ 4] local 10.0.2.4 port 85 connected with 10.0.0.1 port 48505
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.0-12.3 sec  14.0 MBytes  9.56 Mbits/sec

```

```

root@h7b:/home/ubuntu# iperf -c h9a -p 74
-----
Client connecting to h9a, TCP port 74
TCP window size: 85.3 KByte (default)
-----
[ 3] local 10.0.3.2 port 55913 connected with 10.0.0.9 port 74
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-10.5 sec  27.0 MBytes 21.5 Mbits/sec
root@h7b:/home/ubuntu#

```

```

root@h9a:/home/ubuntu# iperf -s -p 74
-----
Server listening on TCP port 74
TCP window size: 85.3 KByte (default)
-----
[ 4] local 10.0.0.9 port 74 connected with 10.0.3.2 port 55913
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.0-11.9 sec  27.0 MBytes 19.1 Mbits/sec

```

Si miramos las tablas de flujos de los switches donde se han configurado las colas QoS, veremos que, aunque desde los controladores sólo se configuran reglas y acciones basadas en puertos de entrada y salida, se añaden las reglas establecidas en los flowspaces de Flowvisor, consiguiendo así el aislamiento entre la administración de los dos tipos de tráfico. En la siguiente imagen se muestra la tabla de flujos del switch S1, donde se puede apreciar lo dicho anteriormente:


```
64 bytes from h4b (10.0.2.4): icmp_seq=24 ttl=64 time=0.162 ms
64 bytes from h4b (10.0.2.4): icmp_seq=25 ttl=64 time=0.217 ms
64 bytes from h4b (10.0.2.4): icmp_seq=26 ttl=64 time=0.186 ms
64 bytes from h4b (10.0.2.4): icmp_seq=27 ttl=64 time=0.136 ms
64 bytes from h4b (10.0.2.4): icmp_seq=38 ttl=64 time=1.15 ms
64 bytes from h4b (10.0.2.4): icmp_seq=39 ttl=64 time=0.145 ms
64 bytes from h4b (10.0.2.4): icmp_seq=40 ttl=64 time=0.153 ms
64 bytes from h4b (10.0.2.4): icmp_seq=41 ttl=64 time=0.142 ms
```

Fig. 25: Conectividad mientras se cambia el camino que sigue el tráfico

Existen más opciones para realizar este cambio y que serían igualmente válidas, como puede ser el establecimiento de nuevas reglas con una prioridad mayor, y en este caso no sería necesaria la eliminación de ninguna regla.

En un escenario convencional de red, hubiera sido bastante más tedioso realizar este cambio, ya que habría que haber configurado individualmente cada uno de los switches implicados en el cambio, lo que hubiera conllevado más tiempo y posiblemente interrupción del servicio. Esta es una de las grandes ventajas del uso de las redes definidas por software, con el aliciente de tener particionada la red con Flowvisor, que asegura que cualquier cambio que realicemos desde uno de los controladores sea transparente al tráfico que se gestiona desde el resto.

CAPÍTULO 4: CONCLUSIONES Y TRABAJOS FUTUROS

Conclusiones

Como primera conclusión es claro que el uso de las arquitecturas SDN se perfila como el futuro a medio plazo en la gestión de las redes, así como los servicios NaaS que bajo este tipo de arquitecturas ya ofrecen a día de hoy algunos fabricantes. Es lógico pensar que esto será así, de hecho en las redes WiFi ya viene ocurriendo desde hace algunos años la existencia de un controlador central donde se establece el plano de control de todos los puntos de acceso dejando a éstos con la funciones del plano de datos.

La idea del particionado de la red, como el que hace Flowvisor u OpenVirteX, puede resultar muy útil en algunos escenarios (no en todos), ya que ofrecen características de delegación, administración y seguridad que en una arquitectura SDN no se tenían. Además, de cara a competir con las actuales técnicas ya implantadas de virtualización de redes (VLAN, MPLS, etc.), la escalabilidad, gestión centralizada, menor coste y mayor facilidad de implementación son los valores más importantes que aportan este tipo de herramientas.

Para la creación de escenarios virtuales se han utilizado a la par dos herramientas como ya hemos visto: Mininet y VNX. Ambas son muy útiles y ambas ofrecen, cada una son API (Python o XML) funciones avanzadas que permiten automatizar bastante la puesta a punto de los escenarios de red.

Existen muchos proyectos en relación con las redes definidas por software que ofrecen características similares pero cada uno de éstos (OpenVirteX, ONOS, OpenDayLight, etc.) se especializa en aportar un punto más de valor a la gestión de las redes: unos particionan la red, otros ofrecen soporte para otros estándares además de Openflow...Lo ideal sería que todas estas ideas se centraran en un único gran proyecto modular donde todas las aportaciones fueran añadiéndose. Aunque no todo el mundo (grandes fabricantes, por ejemplo) está interesado en que se implemente una solución abierta a todos que no sólo mejore la administración de las redes, sino que se haga a menor coste.

Finalmente indicar que las soluciones vistas en este trabajo, Floodlight y Flowvisor, sobre todo esta última, se recomiendan a día de hoy para utilizarlas en entornos de laboratorio y/o de desarrollo, ya que necesitan algo más de madurez de cara a poder presentarse como soluciones lo suficientemente estables para implementarlas en entornos críticos de producción. Como se ha comentado antes, Flowvisor, por ejemplo, no tiene movimiento desde 2013, por lo que parece que si no se retoma este proyecto (o se integra en otros, como Open VirteX), en poco tiempo dejará de ser objetivo de estudio en los trabajos o proyectos de investigación (aunque la idea que implementa Flowvisor no debe olvidarse, ya que ofrece grandes ventajas que deberían seguir mejorando y ampliando, ya sea dentro del mismo proyecto o en otros).

Futuros trabajos

Puesto que el proyecto de Flowvisor no está siendo ya soportado por sus creadores, un claro trabajo que puede aportar bastante valor a las redes SDN es la adaptación de este proyecto a las necesidades actuales, sobretodo en el sentido de la compatibilidad con las nuevas versiones

de Openflow y la implementación de todas las nuevas funcionalidades que han ido aportando desde este estándar.

A día de hoy, y como se ha comentado anteriormente, el proyecto OpenVirteX también está parado y su *roadmap*, según palabras textuales de uno de sus principales desarrolladores, Ali Al-Shabibi, está *congelado*. Por lo que sería interesante no sólo retomar este proyecto sino integrar las funcionalidades de Flowvisor en el mismo, lo cual se une con lo indicado en el primer punto.

Si los dos puntos anteriores se llegaran a implementar, el siguiente paso directo sería la integración de esa futura herramientas y tecnologías, como ONOS o por qué no, OpenStack.

En cuanto a las herramientas de despliegue utilizadas en este proyecto, para que su uso pueda llegar de forma más fácil a cualquier implementación, creo que , la generación de interfaces de usuarios con las que sea posible crear cualquier tipo de escenario sería bueno, y si éstas GUI's se implementaran desde el principio orientadas al usuario final utilizando un lenguaje declarativo, sería bajo mi punto de vista un forma ideal para hacer que éstas herramientas, sobre todo VNX que es la más completa a día de hoy, llegaran a ser utilizadas en muchos más proyectos y entornos tanto de desarrollo como de producción.

Si vamos un paso más allá, estas interfaces de usuarios a las que hacía referencia sería también un buen futuro trabajo el hacerlas extensibles a las arquitecturas SDN (en cuanto a open source se refiere, ya que las soluciones comerciales si cuentan con GUIs) en general, de forma que mediante GUI's con leguajes declarativos, cualquier gestor, sin necesidad de tener conocimientos demasiado técnicos sobre el funcionamiento de las redes, sea capaz de gestionarlas. Por ejemplo, un posible proyecto sería el uso de esta interfaz de usuario para poder gestionar escenarios con Floodlight, Flovisor (actualizado) y como herramienta de despliegue de red, VNX.

Por último, una clara mejora en las arquitecturas SDN sería potenciar la cara de la seguridad (autenticación de acceso a las interfaces de la NB API, por ejemplo). De hecho la implementación de todas las medidas de seguridad en estas nuevas arquitecturas y soluciones deberían nacer de la base, y no ser un añadido una vez están implementadas.

BIBLIOGRAFÍA

- [1] Flowvisor project: <https://github.com/OPENNETWORKINGLAB/flowvisor/wiki>
- [2] Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., & Smeliansky, R. (2013, October). Advanced study of SDN/OpenFlow controllers. In *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia* (p. 1). ACM.
- [3] ONOS Project: <http://onosproject.org/>
- [4] Concepto VLAN: <https://es.wikipedia.org/wiki/VLAN>
- [5] Concepto MPLS: https://es.wikipedia.org/wiki/Multiprotocol_Label_Switching
- [6] Concepto VPN: https://es.wikipedia.org/wiki/Red_privada_virtual
- [7] Concepto VRF: https://en.wikipedia.org/wiki/Virtual_routing_and_forwarding
- [8] ForCES RFCs: <https://datatracker.ietf.org/wg/forces/documents/>
- [9] Openflow en web de Open Networking <https://www.opennetworking.org/sdn-resources/openflow>
- [10] Scott-Hayward, S., O'Callaghan, G., & Sezer, S. (2013, November). Sdn security: A survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for* (pp. 1-7). IEEE.
- [11] Pyretic project web site: <http://www.frenetic-lang.org/pyretic/>
- [12] Ethane Project: <http://yuba.stanford.edu/ethane/>
- [13] ONF Openflow 1.5: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf>
- [14] ONF web page: <https://www.opennetworking.org>
- [15] Openflow 1.0: <http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf>
- [16] Khondoker, R., Zaalouk, A., Marx, R., & Bayarou, K. (2014, January). Feature-based comparison and selection of Software Defined Networking (SDN) controllers. In *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on* (pp. 1-7). IEEE.
- [17] SDXCentral web page: <https://www.sdxcentral.com/>
- [18] Centeno, A. G., Vergel, C. M. R., Calderón, C. A., & Bondarenko, F. C. C. (2014). Controladores SDN, elementos para su selección y evaluación. *Revista Telemática*, 13(3), 10-20.
- [19] Lista de controladores SDN open source : <http://yuba.stanford.edu/~casado/of-sw.html>
- [20] Onos white paper: Introducing ONOS - a SDN network operating system for Service Providers, en: <http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf>
- [21] Beacon controller web: <https://openflow.stanford.edu/display/Beacon/Home>
- [22] Beacon controller study: <http://yuba.stanford.edu/~derickso/docs/hotsdn15-erickson.pdf>
- [23] NOX/-POX: <http://www.noxrepo.org/>
- [24] PyPy entorno de ejecución Python: <http://pypy.org/>
- [25] OpenDayLight Project: <http://www.opendaylight.org/>
- [26] Openflow classification parameters: http://flowgrammable.org/sdn/openflow/classifiers/#tab_ofp_1_0
- [27] Floodlight Project: <http://www.projectfloodlight.org/>
- [28] OpenStack wiki, plugin Neutron: <https://wiki.openstack.org/wiki/Neutron>
- [29] Mininet project : <http://mininet.org/>
- [30] VNX Project: <http://web.dit.upm.es/~vnx>
- [31] Blog simuladores de redes: <http://www.brianlinkletter.com/open-source-network-simulators/>
- [32] Linux network namespaces: <http://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/>
- [33] Virtual Network Design: <http://www.ramonfontes.com/vnd>
- [34] Miniedit: <http://www.brianlinkletter.com/how-to-use-miniedit-mininets-graphical-user-interface/>
- [35] VirtualBox project : <https://www.virtualbox.org/>
- [36] Qemu webpage : <http://www.qemu.org>

[37] Linux containers: <https://linuxcontainers.org/>

[38] Linux bridge : <http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge>

ANEXO I

Configuración utilizada para la resolución del escenario 1 (la configuración de Flowvisor y Floodlight es la utilizada junto con VNX, con Mininet sería similar pero con números de puerto diferentes en las reglas). El directorio bajo el cual se encuentran estos ficheros es *demo0*:

- **Flowvisor** (fichero `flowvisor_fvctl_vpn_lxc.sh`):

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-slice slice1 -p fvadmin
tcp:10.15.0.3:6653 sliceCa@tfm.es
```

```
#Flowspace
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_Sx any
110 any slice1=4
```

- **Floodlight**

- Con módulo de Forwarding no hace falta configurarlo, ya que se el propio controlador envía las reglas de flujos a los switches correspondientes.

- Sin módulo de Forwarding (fichero `floodlight_flow_pusher_sin_fwd.sh`):

```
curl -d '{"switch": "00:00:00:00:00:00:01", "name":"flow-mod-1a",
"cookie":"0", "priority":"42768", "active":"true",
"actions":"output=flood"}'
```

<http://10.15.0.3:8082/wm/staticflowpusher/json>

```
curl -d '{"switch": "00:00:00:00:00:00:02", "name":"flow-mod-2a",
"cookie":"0", "priority":"42768", "active":"true",
"actions":"output=flood"}'
```

<http://10.15.0.3:8082/wm/staticflowpusher/json>

```
curl -d '{"switch": "00:00:00:00:00:00:03", "name":"flow-mod-7a",
"cookie":"0", "priority":"42768", "active":"true",
"actions":"output=flood"}'
```

<http://10.15.0.3:8082/wm/staticflowpusher/json>

- **Mininet** (fichero `topo-vpn.py`):

```
from mininet.topo import Topo
from mininet.node import Host
```

```
class MyTopo( Topo ):
```

```
    def __init__( self ):
        Topo.__init__( self )
```

```
        # Add hosts and switches
```

```
print '*****CREANDO LOS HOSTS*****'
```

```
    h1a = self.addHost( 'h1a',mac='00:00:00:00:00:1a', ip='10.0.0.1/23',
defaultRoute='h1a-eth0' )
```

```
    h2a = self.addHost( 'h2a',mac='00:00:00:00:00:2a', ip='10.0.0.2/23',
defaultRoute='h2a-eth0' )
```

```
    h3a = self.addHost( 'h3a',mac='00:00:00:00:00:3a', ip='10.0.0.3/23',
defaultRoute='h3a-eth0' )
```

```
    h4a = self.addHost( 'h4a',mac='00:00:00:00:00:4a', ip='10.0.0.4/23',
defaultRoute='h4a-eth0' )
```

```
    h5a = self.addHost( 'h5a',mac='00:00:00:00:00:5a', ip='10.0.0.5/23',
defaultRoute='h5a-eth0' )
```

```

        h6a = self.addHost( 'h6a',mac='00:00:00:00:00:6a', ip='10.0.0.6/23',
defaultRoute='h6a-eth0' )
        h7a = self.addHost( 'h7a',mac='00:00:00:00:00:7a', ip='10.0.0.7/23',
defaultRoute='h7a-eth0' )
        h8a = self.addHost( 'h8a',mac='00:00:00:00:00:8a', ip='10.0.0.8/23',
defaultRoute='h8a-eth0' )
        h9a = self.addHost( 'h9a',mac='00:00:00:00:00:9a', ip='10.0.0.9/23',
defaultRoute='h9a-eth0' )
        h10a = self.addHost( 'h10a',mac='00:00:00:00:00:aa', ip='10.0.0.10/23',
defaultRoute='h10a-eth0' )

        h1b = self.addHost( 'h1b',mac='00:00:00:00:00:1b', ip='10.0.2.1/24',
defaultRoute='h1b-eth0' )
        h2b = self.addHost( 'h2b',mac='00:00:00:00:00:2b', ip='10.0.3.1/24',
defaultRoute='h2b-eth0' )

print '*****CREANDO LOS SWITCH*****'

s1 = self.addSwitch( 's1' )
s2 = self.addSwitch( 's2' )
s3 = self.addSwitch( 's3' )

# Add links

print '*****CREANDO LOS LINKS*****'

self.addLink( h1a, s2 )
self.addLink( h2a, s2 )
self.addLink( h3a, s2 )
self.addLink( h4a, s2 )
self.addLink( h5a, s2 )

self.addLink( h6a, s2 )
self.addLink( h7a, s2 )
self.addLink( h8a, s2 )
self.addLink( h9a, s2 )
self.addLink( h10a, s2 )

self.addLink( h1b, s3 )
self.addLink( h2b, s3 )

self.addLink( s1, s2 )
self.addLink( s1, s3 )

print '*****INICIANDO ESCENARIO...*****'
topos = { 'mytopo': ( lambda: MyTopo() ) }

```

- **VNX** (fichero vnx_topo.xml):

```

<vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-2.00.xsd">
<global>
<version>2.0</version>
<scenario_name>Escenario_vpn_demo_1</scenario_name>
<automac/>
<vm_mgmt type="none" />
<!--vm_mgmt type="private" network="10.250.0.0" mask="24" offset="200">
<host_mapping />
</vm_mgmt-->
<vm_defaults>
<console id="0" display="no"/>
<console id="1" display="yes"/>

```

```

    </vm_defaults>
  </global>

  <net name="s2" mode="openvswitch" controller="tcp:10.15.0.2:6633"
of_version="OpenFlow10" hwaddr="00:00:00:00:00:02"/>
  <net name="s3" mode="openvswitch" controller="tcp:10.15.0.2:6633"
of_version="OpenFlow10" hwaddr="00:00:00:00:00:03"/>

  <net name="s1" mode="openvswitch" controller="tcp:10.15.0.2:6633"
of_version="OpenFlow10" hwaddr="00:00:00:00:00:01">
    <connection name='s1-eth1' net='s2'/>
    <connection name='s1-eth2' net='s3'/>
  </net>
  <net name="MgmtNet" mode="virtual_bridge"/>

  <vm name="h1a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
      <ipv4>10.0.0.1/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
  </vm>

  <vm name="h2a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
      <ipv4>10.0.0.2/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
  </vm>

  <vm name="h3a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
      <ipv4>10.0.0.3/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
  </vm>

  <vm name="h4a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
      <ipv4>10.0.0.4/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
  </vm>

  <vm name="h5a" type="lxc">

```

```

    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
        <ipv4>10.0.0.5/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
    <vm name="h6a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
        <ipv4>10.0.0.6/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
    <vm name="h7a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
        <ipv4>10.0.0.7/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
    <vm name="h8a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
        <ipv4>10.0.0.8/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
    <vm name="h9a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
        <ipv4>10.0.0.9/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
    <vm name="h10a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
        <ipv4>10.0.0.10/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>

```

```

    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>

```

```

<vm name="h1b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s3">
    <ipv4>10.0.2.1/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>

```

```

<vm name="h2b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s3">
    <ipv4>10.0.2.2/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>

```

```

<!--FLOWVISOR VM-->
<vm name="Flowvisor" type="lxc">
  <filesystem
type="cow">/usr/share/vnx/filesystems/rootfs_ubuntu</filesystem>
  <if id="1" net="MgmtNet">
    <ipv4>10.15.0.2/24</ipv4>
  </if>
  <filetree seq="on_boot" root="/etc">/etc/flowvisor</filetree>
  <filetree seq="on_boot" root="/etc">/etc/flowvisor.passwd</filetree>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/demo0/vnx/flowvisor_fvc
ctl_vpn_vnx.sh</filetree>
  <exec seq="on_boot" type="verbatim">chown -R flowvisor
/usr/local/share/db/flowvisor</exec>
  <exec seq="on_boot" type="verbatim" user="flowvisor">
  <!--/tmp/queues_vpn.sh-->
    sudo -u flowvisor flowvisor -l /etc/flowvisor/config.json &amp;
    sleep 10;
    /tmp/flowvisor_fvctl_vpn_vnx.sh;
  </exec>
  <exec seq="stop_ctrl" type="verbatim">pkill -f "sudo -u flowvisor
flowvisor"</exec>
</vm>

```

```

<!--FLOODLIGHT VM-->
<vm name="Floodlight_1" type="lxc">
  <filesystem
type="cow">/usr/share/vnx/filesystems/rootfs_ubuntu</filesystem>
  <if id="1" net="MgmtNet">
    <ipv4>10.15.0.3/24</ipv4>
  </if>

```

```
<filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/demo0/vnx/floodlight_flow_pusher_vnx.sh</filetree>
<exec seq="on_boot" type="verbatim">
  cd /etc/controllers/floodlight_forwarding;
  java -jar target/floodlight.jar &
</exec>
<exec seq="stop_ctrl" type="verbatim">pkill -f "java"</exec>
</vm>

<host>
  <hostif net="MgmtNet">
    <ipv4>10.15.0.100/24</ipv4>
  </hostif>
</host>
</vnx>
```

ANEXO II

Configuración utilizada para la resolución del escenario 2 (la configuración de Flowvisor y Floodlight es la utilizada junto con VNX, con Mininet sería similar pero con números de puerto diferentes en las reglas). El directorio bajo el cual se encuentran estos ficheros es *demo1*:

- **Flowvisor**(fichero flowvisor_fvctl_vpn_vnx.sh) :

```
#SLICES

#Creacion
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-slice -p fvadmin
sliceClienteA tcp:10.15.0.3:6653 sliceCa@tfm.es
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-slice -p fvadmin
sliceClienteB tcp:10.15.0.4:6653 sliceCb@tfm.es
#fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-slice -p fvadmin
sliceAdminVPN tcp:127.0.0.1:6655 sliceAdminVpn@tfm.es

#CLIENTE A
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca11
00:00:00:00:00:00:01 110
in_port=1,dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
-q 0,1 -f 0
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca21
00:00:00:00:00:00:02 110
dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca41
00:00:00:00:00:00:04 110
dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca61
00:00:00:00:00:00:06 110
in_port=1,dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
-q 0,1 -f 0

fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca12
00:00:00:00:00:00:01 110
in_port=1,dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
-q 0,1 -f 0
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca22
00:00:00:00:00:00:02 110
dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca42
00:00:00:00:00:00:04 110
dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca62
00:00:00:00:00:00:06 110
in_port=1,dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
-q 0,1 -f 0

fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca13
00:00:00:00:00:00:01 110
in_port=3,dl_type=0x0800,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
-q 0,1 -f 0
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca23
00:00:00:00:00:00:02 110
dl_type=0x0800,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca43
00:00:00:00:00:00:04 110
dl_type=0x0800,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
```



```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca63
00:00:00:00:00:00:00:01 110
in_port=2,dl_type=0x0800,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
-q 0,1 -f 0
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca14
00:00:00:00:00:00:00:01 110
in_port=3,dl_type=0x0806,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
-q 0,1 -f 0
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca24
00:00:00:00:00:00:00:02 110
```

```
dl_type=0x0806,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca44
00:00:00:00:00:00:00:04 110
```

```
dl_type=0x0806,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca64
00:00:00:00:00:00:00:06 110
```

```
in_port=2,dl_type=0x0806,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
-q 0,1 -f 0
```

#CLIENTE B

```
#fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace
fv_fs_cb11test 00:00:00:00:00:00:00:01 110
```

```
dl_type=0x0800,nw_proto=6,nw_src=10.0.2.254,nw_dst=10.0.2.254 sliceClienteB=4 -
q 0,1 -f 1
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb11
00:00:00:00:00:00:00:01 110
```

```
in_port=2,dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.3.0/24 sliceClienteB=4
-q 0,1 -f 1
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb31
00:00:00:00:00:00:00:03 110
```

```
dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.3.0/24 sliceClienteB=4
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb51
00:00:00:00:00:00:00:05 110
```

```
dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.3.0/24 sliceClienteB=4
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb61
00:00:00:00:00:00:00:06 110
```

```
in_port=1,dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.3.0/24 sliceClienteB=4
-q 0,1 -f 1
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb12
00:00:00:00:00:00:00:01 110
```

```
in_port=2,dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.3.0/24 sliceClienteB=4
-q 0,1 -f 1
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb32
00:00:00:00:00:00:00:03 110
```

```
dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.3.0/24 sliceClienteB=4
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb52
00:00:00:00:00:00:00:05 110
```

```
dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.3.0/24 sliceClienteB=4
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb62
00:00:00:00:00:00:00:06 110
```

```
in_port=1,dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.3.0/24 sliceClienteB=4
-q 0,1 -f 1
```

```
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb13
00:00:00:00:00:00:00:01 110
```

```
in_port=3,dl_type=0x0800,nw_src=10.0.3.0/24,nw_dst=10.0.0.0/23 sliceClienteB=4
-q 0,1 -f 1
```

```

fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb33
00:00:00:00:00:00:00:03 110
dl_type=0x0800,nw_src=10.0.3.0/24,nw_dst=10.0.0.0/23 sliceClienteB=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb53
00:00:00:00:00:00:00:05 110
dl_type=0x0800,nw_src=10.0.3.0/24,nw_dst=10.0.0.0/23 sliceClienteB=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb63
00:00:00:00:00:00:00:06 110
in_port=3,dl_type=0x0800,nw_src=10.0.3.0/24,nw_dst=10.0.0.0/23 sliceClienteB=4
-q 0,1 -f 1

fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb14
00:00:00:00:00:00:00:01 110
in_port=3,dl_type=0x0806,nw_src=10.0.3.0/24,nw_dst=10.0.0.0/23 sliceClienteB=4
-q 0,1 -f 1
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb34
00:00:00:00:00:00:00:03 110
dl_type=0x0806,nw_src=10.0.3.0/24,nw_dst=10.0.0.0/23 sliceClienteB=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb54
00:00:00:00:00:00:00:05 110
dl_type=0x0806,nw_src=10.0.3.0/24,nw_dst=10.0.0.0/23 sliceClienteB=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_cb64
00:00:00:00:00:00:00:06 110
in_port=3,dl_type=0x0806,nw_src=10.0.3.0/24,nw_dst=10.0.0.0/23 sliceClienteB=4
-q 0,1 -f 1

```

- **Floodlight:**

RED 1 (fichero Floodlight_flow_pusher_CA.sh):

```

curl -d '{"switch": "00:00:00:00:00:00:00:01", "name":"flow-mod-1a",
"cookie":"0", "priority":"42768", "in_port":"1","active":"true",
"actions":{"output=3"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:01", "name":"flow-mod-2a",
"cookie":"0", "priority":"42768", "in_port":"3","active":"true",
"actions":{"output=1"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:06", "name":"flow-mod-3a",
"cookie":"0", "priority":"42768", "in_port":"2","active":"true",
"actions":{"output=1"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:06", "name":"flow-mod-4a",
"cookie":"0", "priority":"42768", "in_port":"1","active":"true",
"actions":{"output=2"}}' http://10.15.0.3:8082/wm/staticflowpusher/json

curl -d '{"switch": "00:00:00:00:00:00:00:02", "name":"flow-mod-7a",
"cookie":"0", "priority":"42768", "in_port":"1","active":"true",
"actions":{"output=flood"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:02", "name":"flow-mod-8a",
"cookie":"0", "priority":"42768", "in_port":"2","active":"true",
"actions":{"output=flood"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:02", "name":"flow-mod-9a",
"cookie":"0", "priority":"42768", "in_port":"3","active":"true",
"actions":{"output=flood"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:02", "name":"flow-mod-10a",
"cookie":"0", "priority":"42768", "in_port":"4","active":"true",
"actions":{"output=flood"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:02", "name":"flow-mod-11a",
"cookie":"0", "priority":"42768", "in_port":"5","active":"true",
"actions":{"output=flood"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:02", "name":"flow-mod-12a",
"cookie":"0", "priority":"42768", "in_port":"6","active":"true",
"actions":{"output=flood"}}' http://10.15.0.3:8082/wm/staticflowpusher/json

```

```

curl -d '{"switch": "00:00:00:00:00:00:00:04", "name":"flow-mod-13a",
"cookie":"0", "priority":"42768", "in_port":"1","active":"true",
"actions":{"output=flood"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:04", "name":"flow-mod-14a",
"cookie":"0", "priority":"42768", "in_port":"2","active":"true",
"actions":{"output=flood"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:04", "name":"flow-mod-15a",
"cookie":"0", "priority":"42768", "in_port":"3","active":"true",
"actions":{"output=flood"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:04", "name":"flow-mod-16a",
"cookie":"0", "priority":"42768", "in_port":"4","active":"true",
"actions":{"output=flood"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:04", "name":"flow-mod-17a",
"cookie":"0", "priority":"42768", "in_port":"5","active":"true",
"actions":{"output=flood"}}' http://10.15.0.3:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:04", "name":"flow-mod-18a",
"cookie":"0", "priority":"42768", "in_port":"6","active":"true",
"actions":{"output=flood"}}' http://10.15.0.3:8082/wm/staticflowpusher/json

```

- **Mininet** (fichero topo_vpn.py):

```

from mininet.topo import Topo
from mininet.node import Host

```

```

class MyTopo( Topo ):
    def __init__( self ):
        Topo.__init__( self )

        print '*****CREANDO LOS HOSTS*****'

        h1a = self.addHost( 'h1a',mac='00:00:00:00:00:1a', ip='10.0.0.1/23',
defaultRoute='h1a-eth0' )
        h2a = self.addHost( 'h2a',mac='00:00:00:00:00:2a', ip='10.0.0.2/23',
defaultRoute='h2a-eth0' )
        h3a = self.addHost( 'h3a',mac='00:00:00:00:00:3a', ip='10.0.0.3/23',
defaultRoute='h3a-eth0' )
        h4a = self.addHost( 'h4a',mac='00:00:00:00:00:4a', ip='10.0.0.4/23',
defaultRoute='h4a-eth0' )
        h5a = self.addHost( 'h5a',mac='00:00:00:00:00:5a', ip='10.0.0.5/23',
defaultRoute='h5a-eth0' )

        h6a = self.addHost( 'h6a',mac='00:00:00:00:00:6a', ip='10.0.0.6/23',
defaultRoute='h6a-eth0' )
        h7a = self.addHost( 'h7a',mac='00:00:00:00:00:7a', ip='10.0.0.7/23',
defaultRoute='h7a-eth0' )
        h8a = self.addHost( 'h8a',mac='00:00:00:00:00:8a', ip='10.0.0.8/23',
defaultRoute='h8a-eth0' )
        h9a = self.addHost( 'h9a',mac='00:00:00:00:00:9a', ip='10.0.0.9/23',
defaultRoute='h9a-eth0' )
        h10a = self.addHost( 'h10a',mac='00:00:00:00:00:aa', ip='10.0.0.10/23',
defaultRoute='h10a-eth0' )

        h1b = self.addHost( 'h1b',mac='00:00:00:00:00:1b', ip='10.0.2.1/24',
defaultRoute='h1b-eth0' )
        h2b = self.addHost( 'h2b',mac='00:00:00:00:00:2b', ip='10.0.2.2/24',
defaultRoute='h2b-eth0' )
        h3b = self.addHost( 'h3b',mac='00:00:00:00:00:3b', ip='10.0.2.3/24',
defaultRoute='h3b-eth0' )
        h4b = self.addHost( 'h4b',mac='00:00:00:00:00:4b', ip='10.0.2.4/24',
defaultRoute='h4b-eth0' )
        h5b = self.addHost( 'h5b',mac='00:00:00:00:00:5b', ip='10.0.2.5/24',
defaultRoute='h5b-eth0' )

```

```

        h6b = self.addHost( 'h6b',mac='00:00:00:00:00:6b', ip='10.0.3.1/24',
defaultRoute='h6b-eth0' )
        h7b = self.addHost( 'h7b',mac='00:00:00:00:00:7b', ip='10.0.3.2/24',
defaultRoute='h7b-eth0' )
        h8b = self.addHost( 'h8b',mac='00:00:00:00:00:8b', ip='10.0.3.3/24',
defaultRoute='h8b-eth0' )
        h9b = self.addHost( 'h9b',mac='00:00:00:00:00:9b', ip='10.0.3.4/24',
defaultRoute='h9b-eth0' )
        h10b = self.addHost( 'h10b',mac='00:00:00:00:00:ab', ip='10.0.3.5/24',
defaultRoute='h10b-eth0' )

```

```

print '*****CREANDO LOS SWITCH*****'

```

```

s1 = self.addSwitch( 's1' )
s2 = self.addSwitch( 's2' )
s3 = self.addSwitch( 's3' )
s4 = self.addSwitch( 's4' )
s5 = self.addSwitch( 's5' )
s6 = self.addSwitch( 's6' )

```

```

# Add links

```

```

print '*****CREANDO LOS LINKS*****'

```

```

self.addLink( h1a, s2 )
self.addLink( h2a, s2 )
self.addLink( h3a, s2 )
self.addLink( h4b, s2 )
self.addLink( h5b, s2 )

```

```

self.addLink( h6a, s3 )
self.addLink( h7a, s3 )
self.addLink( h8a, s3 )
self.addLink( h9a, s3 )
self.addLink( h10a, s3 )

```

```

self.addLink( h1b, s4 )
self.addLink( h2b, s4 )
self.addLink( h3b, s4 )
self.addLink( h4b, s4 )
self.addLink( h5b, s4 )

```

```

self.addLink( h6b, s5 )
self.addLink( h7b, s5 )
self.addLink( h8b, s5 )
self.addLink( h9b, s5 )
self.addLink( h10b, s5 )

```

```

self.addLink( s1, s2 )
self.addLink( s1, s3 )
self.addLink( s1, s6 )
self.addLink( s6, s4 )
self.addLink( s6, s5 )

```

```

print '*****INICIANDO ESCENARIO...*****'

```

```

topos = { 'mytopo': ( lambda: MyTopo() ) }

```

- **VNX** (fichero vnx_topo.xml):

```

<vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-2.00.xsd">
  <global>
    <version>2.0</version>
    <scenario_name>Escenario_vpn_demo_1</scenario_name>
    <automac/>
    <vm_mgmt type="none" />
    <!--vm_mgmt type="private" network="10.250.0.0" mask="24" offset="200">
      <host_mapping />
    </vm_mgmt-->
    <vm_defaults>
      <console id="0" display="no"/>
      <console id="1" display="yes"/>
    </vm_defaults>
  </global>
  <net name="s2" mode="openvswitch" controller="tcp:10.15.0.2:6633"
of_version="OpenFlow10" hwaddr="00:00:00:00:00:02"/>
  <net name="s3" mode="openvswitch" controller="tcp:10.15.0.2:6633"
of_version="OpenFlow10" hwaddr="00:00:00:00:00:03"/>

  <net name="s1" mode="openvswitch" controller="tcp:10.15.0.2:6633"
of_version="OpenFlow10" hwaddr="00:00:00:00:00:01">
    <connection name='s1-eth1' net='s2' />
    <connection name='s1-eth2' net='s3' />
    <connection name='s1-eth3' net='s6' />
  </net>
  <net name="s6" mode="openvswitch" controller="tcp:10.15.0.2:6633"
of_version="OpenFlow10" hwaddr="00:00:00:00:00:06">
    <connection name='s6-eth2' net='s4' />
    <connection name='s6-eth3' net='s5' />
  </net>

  <net name="s4" mode="openvswitch" controller="tcp:10.15.0.2:6633"
of_version="OpenFlow10" hwaddr="00:00:00:00:00:04"/>
  <net name="s5" mode="openvswitch" controller="tcp:10.15.0.2:6633"
of_version="OpenFlow10" hwaddr="00:00:00:00:00:05"/>

  <net name="MgmtNet" mode="virtual_bridge"/>
  <vm name="h1a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
      <ipv4>10.0.0.1/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
  </vm>
  <vm name="h2a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
      <ipv4>10.0.0.2/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>

```

```

</vm>
<vm name="h3a" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s2">
    <ipv4>10.0.0.3/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h4a" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s2">
    <ipv4>10.0.0.4/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h5a" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s2">
    <ipv4>10.0.0.5/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h6a" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s3">
    <ipv4>10.0.0.6/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h7a" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s3">
    <ipv4>10.0.0.7/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h8a" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s3">
    <ipv4>10.0.0.8/24</ipv4>
  </if>

```

```

    <filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h9a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s3">
        <ipv4>10.0.0.9/24</ipv4>
    </if>
    <filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h10a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s3">
        <ipv4>10.0.0.10/24</ipv4>
    </if>
    <filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h1b" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s4">
        <ipv4>10.0.2.1/24</ipv4>
    </if>
    <filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h2b" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s4">
        <ipv4>10.0.2.2/24</ipv4>
    </if>
    <filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h3b" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s4">
        <ipv4>10.0.2.3/24</ipv4>
    </if>
    <filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>

```

```

<vm name="h4b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s4">
    <ipv4>10.0.2.4/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h5b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s4">
    <ipv4>10.0.2.5/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>

<vm name="h6b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s5">
    <ipv4>10.0.3.1/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h7b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s5">
    <ipv4>10.0.3.2/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h8b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s5">
    <ipv4>10.0.3.3/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h9b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s5">
    <ipv4>10.0.3.4/24</ipv4>
  </if>

```



```

        <filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
        <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
        <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
    </vm>
    <vm name="h10b" type="lxc">
        <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
        <if id="1" net="s5">
            <ipv4>10.0.3.5/24</ipv4>
        </if>
        <filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
        <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
        <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
    </vm>
    <!-- FLOWVISOR VM-->
    <vm name="Flowvisor" type="lxc">
        <filesystem
type="cow">/usr/share/vnx/filesystems/rootfs_ubuntu</filesystem>
        <if id="1" net="MgmtNet">
            <ipv4>10.15.0.2/24</ipv4>
        </if>
        <filetree seq="on_boot" root="/etc">/etc/flowvisor</filetree>
        <filetree seq="on_boot" root="/etc">/etc/flowvisor.passwd</filetree>
        <filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/demo1/vnx/flowvisor_fvc
ctl_vpn_vnx.sh</filetree>
        <filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/demo1/vnx/queues_vpn.sh
</filetree>
        <exec seq="on_boot" type="verbatim">chown -R flowvisor
/usr/local/share/db/flowvisor</exec>
        <exec seq="on_boot" type="verbatim" user="flowvisor">
            <!--/tmp/queues_vpn.sh-->
            sudo -u flowvisor flowvisor -l /etc/flowvisor/config.json &
            sleep 10;
            /tmp/flowvisor_fvctl_vpn_vnx.sh;
        </exec>
        <exec seq="stop_ctrl" type="verbatim">pkill -f "sudo -u flowvisor
flowvisor"</exec>
    </vm>

    <!-- FLOODLIGHT VMs-->
    <vm name="Floodlight_1" type="lxc">
        <filesystem
type="cow">/usr/share/vnx/filesystems/rootfs_ubuntu</filesystem>
        <if id="1" net="MgmtNet">
            <ipv4>10.15.0.3/24</ipv4>
        </if>
        <filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/demo1/vnx/floodlight_fl
ow_pusher_CA.sh</filetree>
        <exec seq="on_boot" type="verbatim">
            cd /etc/controllers/floodlight;
            java -jar target/floodlight.jar &
        </exec>
        <exec seq="stop_ctrl" type="verbatim">pkill -f "java"</exec>
    </vm>

```

```

<vm name="Floodlight_2" type="lxc">
  <filesystem
type="cow">/usr/share/vnx/filesystems/rootfs_ubuntu</filesystem>
  <if id="1" net="MgmtNet">
    <ipv4>10.15.0.4/24</ipv4>
  </if>
  <filetree                               seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/demo1/vnx/floodlight_fl
ow_pusher_CB.sh</filetree>
  <exec seq="on_boot" type="verbatim">
    cd /etc/controllers/floodlight;
    java -jar target/floodlight.jar &
  </exec>
  <exec seq="stop_ctrl" type="verbatim">pkill -f "java"</exec>
</vm>
<host>
  <hostif net="MgmtNet">
    <ipv4>10.15.0.100/24</ipv4>
  </hostif>
</host>
</vnx>

```

- **Configuración colas QoS (fichero queues_vpn.sh):**

```

ovs-vsctl -- set port s1-eth1-0 qos=@newqos -- --id=@newqos create qos
type=linux-htb \
queues=0=@q0,1=@q1 -- --id=@q0 create queue other-config:min-rate=1000000 other-
config:max-rate=10000000 -- \
--id=@q1 create queue other-config:min-rate=1000000 other-config:max-
rate=20000000 -- \

```

```

ovs-vsctl -- set port s1-eth2-0 qos=@newqos -- --id=@newqos create qos
type=linux-htb \
queues=0=@q0,1=@q1 -- --id=@q0 create queue other-config:min-rate=1000000 other-
config:max-rate=10000000 -- \
--id=@q1 create queue other-config:min-rate=1000000 other-config:max-
rate=20000000 -- \

```

```

ovs-vsctl -- set port s1-eth3-0 qos=@newqos -- --id=@newqos create qos
type=linux-htb \
queues=0=@q0,1=@q1 -- --id=@q0 create queue other-config:min-rate=1000000 other-
config:max-rate=10000000 -- \
--id=@q1 create queue other-config:min-rate=1000000 other-config:max-
rate=20000000 -- \

```

```

ovs-vsctl -- set port s1-eth3-1 qos=@newqos -- --id=@newqos create qos
type=linux-htb \
queues=0=@q0,1=@q1 -- --id=@q0 create queue other-config:min-rate=1000000 other-
config:max-rate=10000000 -- \
--id=@q1 create queue other-config:min-rate=1000000 other-config:max-
rate=20000000 -- \

```

```

ovs-vsctl -- set port s6-eth2-0 qos=@newqos -- --id=@newqos create qos
type=linux-htb \
queues=0=@q0,1=@q1 -- --id=@q0 create queue other-config:min-rate=1000000 other-
config:max-rate=10000000 -- \
--id=@q1 create queue other-config:min-rate=1000000 other-config:max-
rate=20000000 -- \

```

```

ovs-vsctl -- set port s6-eth3-0 qos=@newqos -- --id=@newqos create qos
type=linux-htb \

```

```
queues=0=@q0,1=@q1 -- --id=@q0 create queue other-config:min-rate=1000000 other-  
config:max-rate=10000000 -- \  
--id=@q1 create queue other-config:min-rate=1000000 other-config:max-  
rate=20000000 -- \  

```

ANEXO III

Configuración utilizada para la resolución del escenario 3 (la configuración de Flowvisor y Floodlight es la utilizada junto con VNX, con Mininet sería similar pero con números de puerto diferentes en las reglas). El directorio bajo el cual se encuentran estos ficheros es *demo2*:

- **Flowvisor** (fichero `flowvisor_fvctl_vnx.sh`):

```
#SLICES

#Creacion
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-slice -p fvadmin
sliceClienteA tcp:10.15.0.3:6653 sliceCa@tfm.es
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-slice -p fvadmin
sliceClienteB tcp:10.15.0.4:6653 sliceCb@tfm.es
#fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-slice -p fvadmin
sliceAdminVPN tcp:127.0.0.1:6655 sliceAdminVpn@tfm.es

#CLIENTE A
#fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace
fv_fs_ca11test 00:00:00:00:00:00:01 110
dl_type=0x0800,nw_proto=6,nw_src=10.0.0.254,nw_dst=10.0.0.254 sliceClienteA=4 -
q 0,1 -f 0
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca11
00:00:00:00:00:00:01 110
in_port=1,dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca21
00:00:00:00:00:00:02 110
dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca211
00:00:00:00:00:00:02 110
dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.0.0/23 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca41
00:00:00:00:00:00:04 110
dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca411
00:00:00:00:00:00:04 110
dl_type=0x0800,nw_src=10.0.2.0/24,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca612
00:00:00:00:00:00:06 110
in_port=1,dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca614
00:00:00:00:00:00:06 110
in_port=4,dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca81
00:00:00:00:00:00:08 110
in_port=1,dl_type=0x0800,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4

fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca12
00:00:00:00:00:00:01 110
in_port=1,dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca22
00:00:00:00:00:00:02 110
dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca221
00:00:00:00:00:00:02 110
dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.0.0/23 sliceClienteA=4
```

```

fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca42
00:00:00:00:00:00:00:04 110
dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca421
00:00:00:00:00:00:00:04 110
dl_type=0x0806,nw_src=10.0.2.0/24,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca622
00:00:00:00:00:00:00:06 110
in_port=1,dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca624
00:00:00:00:00:00:00:06 110
in_port=4,dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca82
00:00:00:00:00:00:00:08 110
in_port=1,dl_type=0x0806,nw_src=10.0.0.0/23,nw_dst=10.0.2.0/24 sliceClienteA=4

fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca13
00:00:00:00:00:00:00:01 110
in_port=2,dl_type=0x0800,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca131
00:00:00:00:00:00:00:01 110
in_port=4,dl_type=0x0800,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca23
00:00:00:00:00:00:00:02 110
dl_type=0x0800,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca43
00:00:00:00:00:00:00:04 110
dl_type=0x0800,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca63
00:00:00:00:00:00:00:06 110
in_port=2,dl_type=0x0800,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca83
00:00:00:00:00:00:00:08 110
in_port=2,dl_type=0x0800,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4

fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca14
00:00:00:00:00:00:00:01 110
in_port=2,dl_type=0x0806,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca141
00:00:00:00:00:00:00:01 110
in_port=4,dl_type=0x0806,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca24
00:00:00:00:00:00:00:02 110
dl_type=0x0806,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca44
00:00:00:00:00:00:00:04 110
dl_type=0x0806,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca64
00:00:00:00:00:00:00:06 110
in_port=2,dl_type=0x0806,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4
fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace fv_fs_ca84
00:00:00:00:00:00:00:08 110
in_port=2,dl_type=0x0806,nw_src=10.0.2.0/24,nw_dst=10.0.0.0/23 sliceClienteA=4

#CLIENTE B
#fvctl -f /etc/flowvisor.passwd -h 10.15.0.2 -p 8081 add-flowspace
fv_fs_cb11test 00:00:00:00:00:00:00:01 110
dl_type=0x0800,nw_proto=6,nw_src=10.0.2.254,nw_dst=10.0.2.254 sliceClienteB=4 -
q 0,1 -f 1

```



```

curl -d '{"switch": "00:00:00:00:00:00:00:05", "name":"flow-mod-15b",
"cookie":"0", "priority":"42768", "in_port":"3","active":"true",
"actions":"output=flood"}' http://10.15.0.4:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:05", "name":"flow-mod-16b",
"cookie":"0", "priority":"42768", "in_port":"4","active":"true",
"actions":"output=flood"}' http://10.15.0.4:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:05", "name":"flow-mod-17b",
"cookie":"0", "priority":"42768", "in_port":"5","active":"true",
"actions":"output=flood"}' http://10.15.0.4:8082/wm/staticflowpusher/json
curl -d '{"switch": "00:00:00:00:00:00:00:05", "name":"flow-mod-18b",
"cookie":"0", "priority":"42768", "in_port":"6","active":"true",
"actions":"output=flood"}' http://10.15.0.4:8082/wm/staticflowpusher/json

```

- **Mininet (fichero topo_vpn.py):**

```

from mininet.topo import Topo
from mininet.node import Host

```

```

class MyTopo( Topo ):
    def __init__( self ):
        Topo.__init__( self )

        print '*****CREANDO LOS HOSTS*****'

        h1a = self.addHost( 'h1a',mac='00:00:00:00:00:1a', ip='10.0.0.1/23',
defaultRoute='h1a-eth0' )
        h2a = self.addHost( 'h2a',mac='00:00:00:00:00:2a', ip='10.0.0.2/23',
defaultRoute='h2a-eth0' )
        h3a = self.addHost( 'h3a',mac='00:00:00:00:00:3a', ip='10.0.0.3/23',
defaultRoute='h3a-eth0' )
        h4a = self.addHost( 'h4a',mac='00:00:00:00:00:4a', ip='10.0.0.4/23',
defaultRoute='h4a-eth0' )
        h5a = self.addHost( 'h5a',mac='00:00:00:00:00:5a', ip='10.0.0.5/23',
defaultRoute='h5a-eth0' )

        h6a = self.addHost( 'h6a',mac='00:00:00:00:00:6a', ip='10.0.0.6/23',
defaultRoute='h6a-eth0' )
        h7a = self.addHost( 'h7a',mac='00:00:00:00:00:7a', ip='10.0.0.7/23',
defaultRoute='h7a-eth0' )
        h8a = self.addHost( 'h8a',mac='00:00:00:00:00:8a', ip='10.0.0.8/23',
defaultRoute='h8a-eth0' )
        h9a = self.addHost( 'h9a',mac='00:00:00:00:00:9a', ip='10.0.0.9/23',
defaultRoute='h9a-eth0' )
        h10a = self.addHost( 'h10a',mac='00:00:00:00:00:aa', ip='10.0.0.10/23',
defaultRoute='h10a-eth0' )

        h1b = self.addHost( 'h1b',mac='00:00:00:00:00:1b', ip='10.0.2.1/24',
defaultRoute='h1b-eth0' )
        h2b = self.addHost( 'h2b',mac='00:00:00:00:00:2b', ip='10.0.2.2/24',
defaultRoute='h2b-eth0' )
        h3b = self.addHost( 'h3b',mac='00:00:00:00:00:3b', ip='10.0.2.3/24',
defaultRoute='h3b-eth0' )
        h4b = self.addHost( 'h4b',mac='00:00:00:00:00:4b', ip='10.0.2.4/24',
defaultRoute='h4b-eth0' )
        h5b = self.addHost( 'h5b',mac='00:00:00:00:00:5b', ip='10.0.2.5/24',
defaultRoute='h5b-eth0' )

        h6b = self.addHost( 'h6b',mac='00:00:00:00:00:6b', ip='10.0.3.1/24',
defaultRoute='h6b-eth0' )
        h7b = self.addHost( 'h7b',mac='00:00:00:00:00:7b', ip='10.0.3.2/24',
defaultRoute='h7b-eth0' )
        h8b = self.addHost( 'h8b',mac='00:00:00:00:00:8b', ip='10.0.3.3/24',
defaultRoute='h8b-eth0' )

```

```
h9b = self.addHost( 'h9b',mac='00:00:00:00:00:9b', ip='10.0.3.4/24',
defaultRoute='h9b-eth0' )
h10b = self.addHost( 'h10b',mac='00:00:00:00:00:ab', ip='10.0.3.5/24',
defaultRoute='h10b-eth0' )
```

```
print '*****CREANDO LOS SWITCH*****'
```

```
s1 = self.addSwitch( 's1' )
s2 = self.addSwitch( 's2' )
s3 = self.addSwitch( 's3' )
s4 = self.addSwitch( 's4' )
s5 = self.addSwitch( 's5' )
s6 = self.addSwitch( 's6' )
s7 = self.addSwitch( 's7' )
s8 = self.addSwitch( 's8' )
```

```
print '*****CREANDO LOS LINKS*****'
```

```
self.addLink( h1a, s2 )
self.addLink( h2a, s2 )
self.addLink( h3a, s2 )
self.addLink( h4b, s2 )
self.addLink( h5b, s2 )
```

```
self.addLink( h6a, s3 )
self.addLink( h7a, s3 )
self.addLink( h8a, s3 )
self.addLink( h9a, s3 )
self.addLink( h10a, s3 )
```

```
self.addLink( h1b, s4 )
self.addLink( h2b, s4 )
self.addLink( h3b, s4 )
self.addLink( h4b, s4 )
self.addLink( h5b, s4 )
```

```
self.addLink( h6b, s5 )
self.addLink( h7b, s5 )
self.addLink( h8b, s5 )
self.addLink( h9b, s5 )
self.addLink( h10b, s5 )
```

```
self.addLink( s6, s4 )
self.addLink( s8, s5 )
self.addLink( s7, s3 )
self.addLink( s1, s2 )
```

```
self.addLink( s1, s6 )
self.addLink( s1, s8 )
self.addLink( s1, s7 )
```

```
self.addLink( s7, s6 )
self.addLink( s7, s8 )
```

```
self.addLink( s6, s8 )
```

```
print '*****INICIANDO ESCENARIO...*****'
```

```
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

- **VNX** (fichero vxn_topo.xml):

```

<vx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-2.00.xsd">
  <global>
    <version>2.0</version>
    <scenario_name>Escenario_demo_2</scenario_name>
    <automac/>
    <vm_mgmt type="none" />
    <vm_defaults>
      <console id="0" display="no"/>
      <console id="1" display="yes"/>
    </vm_defaults>
  </global>
  <net      name="s2"      mode="openvswitch"      fail_mode="secure"
controller="tcp:10.15.0.2:6633"      of_version="OpenFlow10"
hwaddr="00:00:00:00:00:02"/>
  <net      name="s3"      mode="openvswitch"      fail_mode="secure"
controller="tcp:10.15.0.2:6633"      of_version="OpenFlow10"
hwaddr="00:00:00:00:00:03"/>
  <net      name="s1"      mode="openvswitch"      fail_mode="secure"
controller="tcp:10.15.0.2:6633"      of_version="OpenFlow10"
hwaddr="00:00:00:00:00:01">
    <connection name='s1-eth1' net='s2' />
    <connection name='s1-eth2' net='s6' />
    <connection name='s1-eth3' net='s7' />
    <connection name='s1-eth4' net='s8' />
  </net>
  <net      name="s6"      mode="openvswitch"      fail_mode="secure"
controller="tcp:10.15.0.2:6633"      of_version="OpenFlow10"
hwaddr="00:00:00:00:00:06">
    <connection name='s6-eth2' net='s4' />
    <connection name='s6-eth3' net='s7' />
    <connection name='s6-eth4' net='s8' />
  </net>
  <net      name="s7"      mode="openvswitch"      fail_mode="secure"
controller="tcp:10.15.0.2:6633"      of_version="OpenFlow10"
hwaddr="00:00:00:00:00:07">
    <connection name='s7-eth3' net='s3' />
    <connection name='s7-eth4' net='s8' />
  </net>
  <net      name="s8"      mode="openvswitch"      fail_mode="secure"
controller="tcp:10.15.0.2:6633"      of_version="OpenFlow10"
hwaddr="00:00:00:00:00:08">
    <connection name='s8-eth4' net='s5' />
  </net>
  <net      name="s4"      mode="openvswitch"      fail_mode="secure"
controller="tcp:10.15.0.2:6633"      of_version="OpenFlow10"
hwaddr="00:00:00:00:00:04"/>
  <net      name="s5"      mode="openvswitch"      fail_mode="secure"
controller="tcp:10.15.0.2:6633"      of_version="OpenFlow10"
hwaddr="00:00:00:00:00:05"/>
  <net name="MgmtNet" mode="virtual_bridge"/>
  <vm name="h1a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s2">
      <ipv4>10.0.0.1/24</ipv4>
    </if>
    <filetree      seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  </vm>
</vx>

```

```

    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h2a" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s2">
    <ipv4>10.0.0.2/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h3a" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s2">
    <ipv4>10.0.0.3/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h4a" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s2">
    <ipv4>10.0.0.4/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h5a" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s2">
    <ipv4>10.0.0.5/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h6a" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s3">
    <ipv4>10.0.0.6/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h7a" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s3">

```

```

        <ipv4>10.0.0.7/24</ipv4>
    </if>
    <filetree                                seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h8a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s3">
        <ipv4>10.0.0.8/24</ipv4>
    </if>
    <filetree                                seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h9a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s3">
        <ipv4>10.0.0.9/24</ipv4>
    </if>
    <filetree                                seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h10a" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s3">
        <ipv4>10.0.0.10/24</ipv4>
    </if>
    <filetree                                seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h1b" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s4">
        <ipv4>10.0.2.1/24</ipv4>
    </if>
    <filetree                                seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h2b" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s4">
        <ipv4>10.0.2.2/24</ipv4>
    </if>
    <filetree                                seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>

```

```

    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h3b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s4">
    <ipv4>10.0.2.3/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h4b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s4">
    <ipv4>10.0.2.4/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h5b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s4">
    <ipv4>10.0.2.5/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h6b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s5">
    <ipv4>10.0.3.1/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h7b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s5">
    <ipv4>10.0.3.2/24</ipv4>
  </if>
  <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
  <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
  <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
<vm name="h8b" type="lxc">
  <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
  <if id="1" net="s5">

```

```

        <ipv4>10.0.3.3/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
    <vm name="h9b" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s5">
        <ipv4>10.0.3.4/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
    <vm name="h10b" type="lxc">
    <filesystem type="cow">/usr/share/vnx/filesystems/rootfs_lxc</filesystem>
    <if id="1" net="s5">
        <ipv4>10.0.3.5/24</ipv4>
    </if>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/hosts</filetree>
    <exec seq="on_boot" type="verbatim">cat /tmp/hosts >> /etc/hosts</exec>
    <exec seq="on_boot" type="verbatim">route add -net 0.0.0.0 netmask 0.0.0.0
dev eth1</exec>
</vm>
    <!-- FLOWVISOR VM-->
    <vm name="Flowvisor" type="lxc">
    <filesystem
type="cow">/usr/share/vnx/filesystems/rootfs_ubuntu</filesystem>
    <if id="1" net="MgmtNet">
        <ipv4>10.15.0.2/24</ipv4>
    </if>
    <filetree seq="on_boot" root="/etc">/etc/flowvisor</filetree>
    <filetree seq="on_boot" root="/etc">/etc/flowvisor.passwd</filetree>
    <filetree seq="on_boot"
root="/tmp">/home/administrador/tfm_jcm/escenarios/vpn/demo2/vnx/flowvisor_fvc
tl_vnx.sh</filetree>
    <exec seq="on_boot" type="verbatim">chown -R flowvisor
/usr/local/share/db/flowvisor</exec>
    <exec seq="on_boot" type="verbatim" user="flowvisor">
    <!--/tmp/queues_vpn.sh-->
        sudo -u flowvisor flowvisor -l /etc/flowvisor/config.json &
sleep 10;
        /tmp/flowvisor_fvctl_vnx.sh;
    </exec>
    <exec seq="stop_ctrl" type="verbatim">pkill -f "sudo -u flowvisor
flowvisor"</exec>
</vm>

    <!-- FLOODLIGHT VMs-->
    <vm name="Floodlight_1" type="lxc">
    <filesystem
type="cow">/usr/share/vnx/filesystems/rootfs_ubuntu</filesystem>
    <if id="1" net="MgmtNet">
        <ipv4>10.15.0.3/24</ipv4>
    </if>

```

```
<filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/demo2/vnx/floodlight_fl
ow_pusher_CA.sh</filetree>
  <exec seq="on_boot" type="verbatim">
    cd /etc/controllers/floodlight;
    java -jar target/floodlight.jar &
    sleep 5;
    /tmp/floodlight_flow_pusher_CA.sh
  </exec>
  <exec seq="stop_ctrl" type="verbatim">pkill -f "java"</exec>
</vm>
<vm name="Floodlight_2" type="lxc">
  <filesystem
type="cow"/>/usr/share/vnx/filesystems/rootfs_ubuntu</filesystem>
  <if id="1" net="MgmtNet">
    <ipv4>10.15.0.4/24</ipv4>
  </if>
  <filetree                                seq="on_boot"
root="/tmp"/>/home/administrador/tfm_jcm/escenarios/vpn/demo2/vnx/floodlight_fl
ow_pusher_CB.sh</filetree>
  <exec seq="on_boot" type="verbatim">
    cd /etc/controllers/floodlight;
    java -jar target/floodlight.jar &
    sleep 5;
    /tmp/floodlight_flow_pusher_CB.sh
  </exec>
  <exec seq="stop_ctrl" type="verbatim">pkill -f "java"</exec>
</vm>
<host>
  <hostif net="MgmtNet">
    <ipv4>10.15.0.100/24</ipv4>
  </hostif>
</host>
</vnx>
```